RECOMMENDER SYSTEM FOR AUDIO RECORDINGS

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jong Seo Lee

January 2010

© 2010
Jong Seo Lee
ALL RIGHTS RESERVE

COMMITTEE MEMBERSHIP

TITLE: RECOMMENDER SYSTEM
FOR AUDIO RECORDINGS

AUTHOR: Jong Seo Lee

DATE SUBMITTED: January 2010

COMMITTEE CHAIR: Alexander Dekhtyar, PhD.
Associate Professor, Computer Science Dept.
California Polytechnic State University, San Luis Obispo

COMMITTEE MEMBER: Gene Fisher, PhD.
Professor, Computer Science Dept.
California Polytechnic State University, San Luis Obispo

COMMITTEE MEMBER: Franz Kurfess, PhD.
Professor, Computer Science Dept.
California Polytechnic State University, San Luis Obispo

ABSTRACT

RECOMMENDER SYSTEM FOR AUDIO RECORDINGS

Jong Seo Lee

Nowadays the largest E-commerce or E-service websites offer millions of products for sale. A *Recommender system* is defined as software used by such websites for recommending commercial or noncommercial product items to users according to the users' tastes. In this project, we develop a recommender system for a private multimedia web service company. In particular, we devise three recommendation engines using different data filtering methods – named *weighted-average*, *K-nearest neighbors*, and *item-based* – which are based on collaborative filtering techniques, which work by recording user preferences on items and by anticipating the future likes and dislikes of users by comparing the records, for prediction of user preference. To acquire proper input data for the three engines, we retrieve data from database using three data collection techniques: *active filtering*, *passive filtering*, and *item-based filtering*. For experimental purpose we compare prediction accuracy of those three recommendation engines with the results from each engine and additionally we evaluate the performance of *weighted-average* method using an *empirical analysis* approach – a methodology which was devised for verification of predictive accuracy.

ACKNOWLEDGMENTS

Glory to God the Father, Lord Jesus Christ, and the Holy Spirit!!! They always lead me in their righteousness and make my way straight before my face. I'll praise and glorify them forever. Amen.

I would like to thank my thesis advisor Dr. Alexander Dekhtyar, without whom this document would not exist. Also, my gratitude goes to Mr. Michael Monegan who provided computer server and database environment for the programming portion of this thesis.

In addition, I would like to thank my lovely family – my father Jung Ki, my mother Ki Bong, and my sister Jong Eun, whose advice, providence, and prayers have guided me throughout my life. I also thank my uncle Rev. Sang Hoon Lee and my cousin Dr. David Lee for their mentorship in my study and career. Finally I also thank Miss. Myeong Sook Lee who gave me kind help on polishing this thesis document.

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF EQUATIONS	x
LIST OF DEFINITIONS	xi
CHAPTER 1. Introduction	1
1.1. Recommender System	1
1.2. The Problem	7
1.3. Contributions	9
1.4. Organization	14
1.5. A Note on This Project	15
CHAPTER 2. Background / Related Work	16
2.1. Collaborative Filtering	16
2.2. Related Work	27
CHAPTER 3. Problem & Solution	30
3.1. The Problem	30
3.2. Solution	32

CHAPTER 4. Implementation	51
4.1. System Architecture	51
4.2. System Components	53
CHAPTER 5. Experimental Evaluation	61
5.1. Comparative Analysis	61
5.2. Accuracy Evaluation	65
CHAPTER 6. Conclusion / Future Work	70
6.1. Achievement & Conclusion	70
6.2. Future Work	70
REFERENCES	72

LIST OF TABLES

[Table 1: tbl_user_info]	33
[Table 2: tbl_title]	33
[Table 3: tbl_title_ratings]	34
[Table 4: tbl_title_segment]	34
[Table 5: tbl_visited_segment]	35
[Table 6: tbl_topic_title_map]	35
[Table 7: tbl_userbased]	36
[Table 8: tbl_knearbased]	36
[Table 9: tbl_itembased]	37
[Table 10: SQL Query Table]	38
[Table 11: Rating Values]	39
[Table 12: Condition Table for L_{score}]	44
[Table 13: Variables and Relevant Queries]	44
[Table 14: tbl_title_topic_master]	48
[Table 15: Comparative Analysis Result Table]	63
[Table 16: Accuracy Evaluation Result Table]	68

LIST OF FIGURES

[Figure 1: Working System Architecture]	13
[Figure 2: Utility Matrix for Recommendation ($v[i, j]$)].....	17
[Figure 3: Item-Topic Relation Matrix]	48
[Figure 4: Recommender System Architecture]	52
[Figure 5: User-based Method Architecture]	54
[Figure 6: Item-based Method Architecture]	58
[Figure 7: Chart for Deviation Result]	69

LIST OF EQUATIONS

[Equation 1]	20, 46, 55
[Equation 2]	20
[Equation 3]	20
[Equation 4]	21, 46
[Equation 5]	21, 47, 56
[Equation 6]	41
[Equation 7]	41
[Equation 8]	49, 59
[Equation 9]	67
[Equation 10]	67

LIST OF DEFINITIONS

[Definition 1]	21, 56
[Definition 2]	62

Chapter 1. Introduction

This thesis describes development of a recommender system engine which was implemented for a private audio contents web service company and evaluation of the engine's performance.

1.1. Recommender Systems

A *recommender system* is defined as a system-level software utilizing a *specific type of information-filtering technique* that presents or recommends commercial or non-commercial product items customized to each user's interests [24]. Examples of such products include, but are not limited to, movies, music, books, news, images, and web blogs.

Recommender systems are promising and powerful tools as they offer a smart and personal way to optimize information selection for each user in the day and age of information overload. As web services are expanding and the service types become more varied, information on the web is increasing at an exponential rate. Users can experience difficulty in finding information that meets their needs and interests. In response, the web service and e-business companies are developing their recommender systems to help their customers get better information on the items of their interests [6].

Applications that recommend products to users have become ubiquitous these days. Recommender Systems are regarded as a tool for win-win strategy in web service and e-commerce areas. The users can get benefits by receiving personally customized information about the multimedia which they are likely to buy or enjoy, and thus reducing time spent on searching for those items. At the same time, the business can get

benefits of an increase in sales or promotion [17]. In addition, recommender systems have a potential to provide businesses with information about the users. For example, data collected by a recommender system may show how the preferences and tastes of the public are changing. A supplier may respond to the trend by promoting products and services in order to strengthen the business in the market. Therefore it is anticipated that recommender systems could evolve into highly effective software application and become an important tool for e-service and e-commerce in the near future.

Real World Recommender Systems

Recommender systems are used as components of larger systems as well as the driving engines behind the business models of some companies. We discuss a number of examples of existing on-line recommender systems which illustrate why recommender systems are popular, practical, and promising.

Amazon.com

At Amazon.com, the web service system uses a recommendation system to personalize the online store for each customer. The recommender system of Amazon.com is famous for its item-based filtering with its computational algorithm scalable independently to the number of customers and the number of items [12]. The recommender system of Amazon.com has three information collection parts titled *Your Browsing History*, *Rate These Items*, and *Improve Your Recommendations* on the website. It also has two suggestion parts for users – *Today's Recommendations For You* and *Recommended For You*. In the *Your Browsing History* section of the web site, the system shows the items

the customer recently searched and displays closely-related items as recommendations. In the *Rate These Items* section, customers are asked to rate some product items on which they might have an opinion. In the *Improve Your Recommendations* section, customers are asked to rate product items they have bought through the Amazon.com web site. In the *Today's Recommendations For You* and *Recommended For You* sections, the system suggests items based on the rating scores obtained from both *Rate These Items* and *Improve Your Recommendations* sections. The system's list of recommended items changes as soon as the customer makes or changes a rating; item-based recommendation technique makes real-time update possible.

Another interesting aspect is that the web service system encourages user to organize her social network through the site by running *Your Communities* page, thereby enabling the system to observe the user's activity, such as emails, coupons, friends, interesting people, and personal opinion for products [18]. In so doing, the system can discover the taste of the user and create appropriate item suggestions for her.

Application for iTunes

The Music Recommender System for iTunes is one of the most popular recommender systems. It is an application for iTunes that is used for the integrated rating system, not for music download.

It uses collaborative filtering techniques to generate music recommendations. The system takes ratings from each user's iTunes play lists and compares the ratings with those of other iTunes users who also have rated their own music choices. Based on the information each iTunes user holds, the system can determine the musical tastes of the

user, predict what else the user would like, and recommend potential music items of interest to the user.

YouTube

Another example of a multimedia recommender system appears on the YouTube website. Although it displays many kinds of categories – *Hot Topics*, *Subscriptions*, *Friend Activity*, *Featured Videos*, and *Videos Being Watched Now* on its main page – that could attract interest of users, only the *Recommended for You* feature works with a recommendation algorithm.

The recommendation list of *Recommended for You* changes its contents as soon as the user visits any video content. Recommendations are also featured on each video play page under the title “*Related Videos*”. The suggestion list that is displayed on each video is fixed; that is, every time the user watches a same video, the same list of videos is displayed. It means that the web service system uses a pre-made suggestion list, and the user preferences and activity do not influence much on the formation of the suggestion list. Such prompt reaction with pre-made suggestion list is possible only with an item-based method. Therefore we can easily perceive that the recommendation lists in the *Recommended for You* and *Related Videos* sections are created by using an item-based method. More details about the item-based method will be discussed in Section 2.1.4.3. The static mechanism of an item-based method is potentially desirable in considering the characteristic of video contents because users tend to search video contents which are closely related one another. For example, suppose a user searches a movie by typing its name. Usually, as a movie item which has long running hours is divided and saved

consecutively in storage. In this case, a suggestion list that includes the remaining parts of the movie in succession is more helpful for the user than other types of suggestion. For another example, if a user searches a music video by typing its singer's name, a suggestion list of songs sung by the singer will likely be a good suggestion.

In these two examples, we can assume that searching by a keyword like movie name or singer's name is used as a correlation factor for item-based method. Using a proper correlation factor is very important for an item-based method to generate efficient recommendation list.

Last.fm

Last.fm is an Internet radio and music community website. Users can access the Last.fm website and provide personal preferences such as name of song or music she wants to listen. In response, the website selects songs or music that are a very good match for the user's taste and plays those continuously; this behavior is similar to that of an FM radio. The website uses "Audioscrobbler", a software application for music recommendation. To "scrobble" songs means that the software application installed in user's computer records information about every piece of music that its user listens to and builds a detailed profile of the user's musical taste and preferences [14]. Then the software application sends the user data to the Last.fm database and Last.fm's recommender system comes up with recommendations based on the data. This process is called "automatic track logging scrobbling". The system adopts a collaborative filtering method for recommendation list creation and it provides a user a list of artists which other people with similar tastes have on their profiles [22].

Last.fm also offers ways for the users to organize their social network through the site like Amazon.com. The system allows the formation of user groups between users with common musical tastes and offers many kinds of services like group discussion, board activity, and group member's individual music suggestion for other people in the group.

Pandora Radio

Pandora radio is another music supply website very similar to Last.fm. Pandora, main software of the website, is a kind of automated music recommendation system that enables users to listen to songs they are likely to enjoy. Pandora radio originated from the Music Genome Project – a comprehensive analysis of music in which a group of musician-analysts listens to and studies each song and describes and notes nearly 400 attributes of each [23]. Pandora provides this information to the public. On the Pandora website, once a user enters a song or artist that she likes, Pandora makes a “station” with the name of the song or artist and plays selections that are musically similar to the entered song or songs of the entered artist.

In contrast to Last.fm, which makes users' social network and thus depends on the correlation between users, Pandora relies on a concept of rich metadata to find relationships between individual songs. To intensify such functional characteristics, Pandora allows users to control the kind of music that is played. While a song is played, users can give feedback – whether they like the song or not – to Pandora. The feedback determines if the song should be played, and how much should similarly classified songs be played on the station [23]; if Pandora becomes aware that the user doesn't like the

song from the feedback, it stops playing the current song, excludes the song from the recommendation list, and plays other recommendable songs.

As a measure of promoting online music items sales, the music broadcast on Pandora radio is also available for purchase on Amazon.com, iTunes Store and other Internet websites.

1.2. The Problem

This project started in conjunction with a start-up company whose business was to operate an on-line subscription service for audio recordings. The company had already created an audio recording delivery web service including the supporting database system. At the onset of this project, the company was looking to add a recommendation tool for users of their web service. This thesis describes the organization of the recommender system for the company's web service software.

The company provided its users subscription services to the audio recordings of various self-help and self-improvement materials – authors' reading their books and recorded lectures etc. – in various topics ranging from personal relationships to homemaking, career, and investment. Users who sign up to such a service exhibit patterns of behavior different from users of other multimedia delivery services (e.g., iTunes, YouTube). We detected some users' tendencies specific to the company's business as follows:

- Usually users do not listen to same self-help audio recording as many times as they repeat listening to the same music.
- Users tend to stick to audios closely related to their favorite topics.

- By and large the running time of audio recordings is longer than music and video clips, but shorter than full movies.
- Users are not as enthusiastic about audio recordings as they are passionate about music.
- Users do not search for alternative or replacing items when there are not items of their interest.

Given these distinctive characteristics of the company's business and the behavior of its users, we wanted to know which method of the two representative data filtering methods – user-based collaborative filtering and item-based filtering – would be more effective for creating recommendations for users. With an expectation that user-based collaborative filtering would perform better than item-based filtering, we tried to implement three data filtering engines. The key concept of the two engines was derived from the user-based filtering method; and the key concept of the other was from the item-based filtering method. After the engines were implemented, we worked on comparison and evaluation to find out which one is better for the audio recordings service.

Title and Segment

To maintain the organization of audio recording items, the company's web service system keeps every audio recording as a segment under a specific title. The title is distinguished by its title id and each title can have many numbers of segments. Therefore segments that are included in a same title have common topics. To score items, users can give rating scores on titles, not on individual segment.

User Rating

In general the most important resource for recommender systems is a database documenting user activity, and this project was no exception to this rule. The company's accumulated data on *users*, *titles*, *subscriptions*, *ratings*, and *user behavior* are an indispensable source in this project. These data need to be processed to make fit for use in the recommender system; for this reason the system collects unrefined data from the database and converts the data to usable data types. We call such unrefined data “*raw* data”; and the usable data “*user rating* data”. The *user rating* data has an adequate data type for the system and is used as direct input for the data filtering engines. The conversion process from *raw* data to *user rating* data can be designed in various ways by its system designer. In this project, to establish *user rating* data set, we import two types of *raw* data: user's direct rating score on audio recording items and data reflecting user's listening patterns. A brief design concept of conversion process is introduced in Section 1.3.1.1.

1.3. Contributions

This thesis offers three main contributions: model development, working system, and analysis and evaluation. Each contribution is discussed in turn below.

1.3.1. Model Development

We designed a special-purpose recommendation model for the recommendation of self-help audio materials based on the available data.

1.3.1.1. Modeling User Ratings

To construct a model of *user ratings* data, we developed a regular and systematic way of processing the *raw* data. As mentioned in Section 1.2., there are two kinds of *raw* data sources: direct user ratings on items and implicit data from users' listening patterns. Direct user ratings are stored in the web service system's database and can be directly used for preparing recommendations. However, there is a tendency that the web service system might have only a limited number of direct user ratings.

The amount of information about user listening patterns is significantly larger. However, to establish *user rating* data from implicit data from a user's listening patterns, additional preprocessing steps are required. We developed three measures and a condition table to set up *user rating* data from implicit data of listening patterns. We modeled the implicit *user rating* as follows:

- 1) *Raw* data which discloses how often and how recently a user visited an audio recording item are collected from the web service system's database.
- 2) The recommender system makes use of those *raw* data to estimate how much interest a user shows in an audio recording title.
- 3) Then, based on our estimates, the recommender system assesses how much a user likes an audio recording title and decides *user rating* data values according to the model we have proposed.

The details of *user rating* modeling are discussed in Section 3.2.3.1.

1.3.1.2. Prediction Calculation Engines

We equipped this recommender system with three different core engines which generate predicted *user rating* data. Based on such predicted *user rating* data on items, the recommender system creates recommendation lists.

The first engine uses the *weighted-average method* whose brief steps are as follows:

- The engine determines the similarities of tastes between a user and all the other users.
- Then based on the similarity data the engine predicts which items would be of interest to the user.

For detailed design, we surveyed some statistical and mathematical theories related to data filtering and after that we concluded that *Pearson Correlation Coefficient* – a useful method for detecting the similarity between data – and *memory-based algorithm* – a mathematic function utilizing the past records for prediction – are proper fits for the two steps of the *weighted-average method* above [3]. Accordingly, *Pearson Correlation Coefficient* and *memory-based algorithm* were taken up as key components for the *weighted-average method*.

The second engine uses the *K-nearest neighbors method* which has the same mechanism as the *weighted-average method* except the fact that the *K-nearest neighbors* method works based on correlations between a user and *K-nearest* users, not all other users [16].

The third engine uses the *item-based method* which works based on correlations between items, instead of users.

Both the second and the third engines were implemented for the purpose of performance comparison with the *weighted-average method*.

1.3.2. Working System

We designed and implemented our recommender system to interact with the company's web service system. The software model was developed in the company's system environment. By using JDBC, we connected the recommender system to the company's MySQL database server to make it interact with the web service system's database. The recommender system collects data from the database, converts the *raw* data to *user rating* data, calculates predictions using *user rating* data, and produces results – recommendation lists; all these jobs are executed during an offline process that is intended to run nightly. After that, the results from recommender system are stored and maintained in the database that can be used to provide necessary recommendation lists for real time requests of the web service system.

Figure 1 shows the architecture of our developed recommender system. The *Raw Data Storage* component offers temporary data storage for the *raw* data. The *User Rating Generation* component converts *raw* data into *user rating* data. After that, the *user rating* data are supplied to three prediction calculation parts, *Weighted Average Method*, *K-nearest Neighbors Method*, and *Item-based Method*.

After those three components finish calculations of the prediction, the results are sent to both the *Database for Results* – which archives the results – and *Result Comparison* – which analyzes the results and compares the performance of those engines. Detailed recommender system architecture design will be discussed in Chapter 4.

[Figure 1: Working System Architecture]

1.3.3. Analysis and Evaluation

We analyzed the performance of the three recommendation engines as follows. As mentioned above, to test the performance of the *weighted-average method* engine, we implemented two additional prediction engines – *K-nearest neighbors* and *item-based method* engines – and also devised software which compares each engine’s prediction result and actual user purchase result and counts how many accurate prediction each recommendation engine made.

Additionally, for the evaluation of the *weighted-average method* engine, we included an *empirical analysis* approach which was devised for the verification of predictive accuracy [3]. Details of comparison and evaluation will be discussed in Chapter 5.

1.4. Organization

The rest of the thesis is organized as follows. Chapter 2 provides an overview of some background and related work. Chapter 3 describes the design of the recommender system. Chapter 4 explains in detail how we implemented the recommender system software. Chapter 5 describes how we analyzed the working performance of the software tools and evaluated the three collaborative methods. Chapter 6 contains conclusion and describes future work.

1.5. A Note on This Project

At the beginning, the main purpose of this project was implementation of recommendation engine using *collaborative filtering techniques* and the engine was planned to be integrated with the company's web service software suite. We started to develop a commercial recommender system for the company, but unexpectedly, they changed the business model because of internal problems, and the plan for a commercial recommender system was cancelled.

Despite the change in the company's business, we were able to use the database system of the company. Although we only had a limited amount of data coming from the actual use of the system because the web site was no longer actively functioning, we recommitted ourselves to adopt three different data filtering methods, implement recommendation engines for those filtering methods, and work on comparison and evaluation of those engines. Thereby the recommender system was designed and implemented independently with the web service systems in the company.

Chapter 2. Background / Related Work

In this chapter we discuss collaborative filtering methods used to produce recommendations in recommender systems and describe some related work. Section 2.1. is devoted to the collaborative filtering methods. In Section 2.2., related work is discussed.

2.1. Collaborative Filtering

The term “Information filtering” refers to a collection of methods, algorithms and techniques for selecting useful or necessary information from a large information streams or data sets [21]. “Collaborative filtering” is a variation of information filtering. The underlying assumption of collaborative filtering is that *people who agreed on many things in the past tend to agree on other things again in the future* [20]. Collaborative filtering is performed by recording user preferences on items they experienced [7], and predicting the future likes and dislikes of users by comparing their past records to each other. We discuss the basics of collaborative filtering below.

2.1.1. The Problem

In an environment where there is a set of users and their preferences over a set of items, collaborative filtering applies statistical techniques and, for each user, finds new items for which they would have high preferences.

We introduce the following notation:

- Let $\{ U_1, U_2, U_3, \dots, U_i, \dots, U_M \}$ be users and i be id of each user.
- Let $\{ I_1, I_2, I_3, \dots, I_j, \dots, I_N \}$ be all possible items and j be id of each item.

- $v_{i,j}$ is a utility function which is a rating that user i gives to item j .
- $v_{i,j}$ is a partial function, i.e., not every user-item pair has a known rating.
- $v_{i,j}$ can also be presented as a *utility matrix* $v[i,j]$, where $v[i,j] = v_{i,j}$. Figure 2 shows an example of the *utility matrix*.

In most situations the users don't rate all the items. Therefore the utility function is incomplete. In most cases the matrix $v[i,j]$ is sparse. An essential condition of making appropriate recommendations is to estimate proper rating values for the unknown – shown as “ $\emptyset$ ” in Figure 2 – values.

		→						
Item id (j)		1	2	3	4	5	6	7
User id (i)		↓						
1		5	4	1	1	5	3	2
2		3	1	4	4	$\emptyset$	5	5
3		3	1	1	4	5	$\emptyset$	10
4		3	1	3	4	1	$\emptyset$	2
5		1	5	5	2	..	..	..

[Figure 2: Utility Matrix for Recommendation ($v[i,j]$)]

Two main problems need to be solved in this situation:

- Estimation of ratings for items not yet rated by a specific user.

- Making recommendations for the user by selecting the items with the highest ratings among all the estimated ratings.

The goal of collaborative filtering is to solve the two main problems by:

- Predicting $v_{i,j}$ for all the pairs of (i, j) for which v is not specified.
- For each user i , finding the top k items $\{ I'_1, \dots, I'_p, \dots, I'_k \}$ which the user i has not rated on and which we have the highest prediction scores.
- For each item j , finding the top m users who will most likely to prefer item j $\{ U'_1, \dots, U'_q, \dots, U'_m \}$ among the users who have not rated the item.

2.1.2. Memory-based vs. Model-based methods

There are two general classes of collaborative filtering methods: memory-based and model-based methods [3]. Memory-based collaborative methods essentially are heuristics that compute rating predictions based on the entire collection of existing ratings [1]. In detail, these methods maintain a database of all user preferences for all items, and make predictions – how much every user will like every item which the user has not rated – by computation across all related data records; for the computation, the algorithm in this category employs statistical techniques to combine users' preferences on items to produce predictions [16]. Model-based collaborative methods develop a model of user ratings to provide rating predictions [16] based on the premise that user preferences conform to a specific model; therefore these methods use the data available to compute the parameters of the model. The model in this context is a prediction function which takes a probabilistic approach [1]. The simple process is: at first, the users' preferences

are accumulated into a model and an underlying model of user preferences is constructed; after that, the model is used for making predictions and generation of recommendations.

Generally speaking memory-based methods are easier to apply, and they produce pretty good prediction results and also have the flexibility new data can be added easily and incrementally. However, this approach can face scalability problems – in terms of both time and space complexity – as the size of the data increases. Moreover, with these methods, the software system cannot provide further information to discern the true nature of data. In model-based methods, although the time complexity to accumulate the data into a model may consume significant time and adding a new data point may require repeating the entire process, once the model (prediction function) is constructed, predictions can be calculated quickly.

2.1.3. Memory-based methods

In this project we adopted Memory-based methods because of their simplicity and practical advantages mentioned in Section 2.1.2. The memory-based algorithm utilizes statistical methods to find a set of users who have similar history of tastes with the active user [16] as follows.

2.1.3.1. Weighted Average Method

This method is one of the most common prediction methods. The basic concept is that we compare the preferences of one user (represented by a) to all the other users (represented by i , where $a \neq i$). Then, with the comparison result, the method can predict user a 's ratings for each unrated item and suggest unrated items with the highest

scores to the user. In case the user a has not rated item j , the predicted rating score of the user a for item j , $P_{a,j}$, is represented as [3]:

$$P_{a,j} = \bar{v}_a + k \sum_{i=1}^n w(a,i)(v_{i,j} - \bar{v}_i) \quad [\text{Equation 1}]$$

Terms of Equation 1 are defined as follows:

$w(a,i)$: the similarity coefficient between user a and user i .

k : normalizing factor [1], selected as

$$k = 1 / \sum_{i=1}^n |w(a,i)| \quad (\text{where } a \neq i) \quad [\text{Equation 2}]$$

$\bar{v}_a$: average of all the ratings of user a

$\bar{v}_i$: average of all the ratings of user i and represented as Equation 3 [3]:

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j} \quad [\text{Equation 3}]$$

Terms of Equation 3 are defined as follows:

I_i : the set of items on which user i has voted

2.1.3.2. Pearson Correlation Coefficient

The similarity coefficient, $w(a,i)$, shown in Equation 1, is a *weight* value presenting how much the tastes of user a and user i are similar. For the calculation of the *weight*, we adopted the *Pearson Correlation Coefficient* method which is traditionally used for the similarity measure in collaborative filtering methods since its first use by GroupLens, one of the first collaborative filtering projects [10]. The *weight* between two users is

based on their ratings for items that both users have rated. The formula of the weight is defined as [3] [5]:

$$w(a, i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \quad [\text{Equation 4}]$$

2.1.3.3. K - Nearest Neighbors Method

The K -Nearest Neighbors, or KNN , method differs from the *weighted average method* in that it uses information from only K users who are most similar in their ratings to the user for whom the predictions are constructed. This method compares the preferences between one user (represented by a) and the K -Nearest users who are represented by i (where $i \in \{i \mid \text{rank}(w(a, i)) \leq K \text{ and } a \neq i\}$) when executing the Equation 1. This method can predict the rating scores of a user's unrated items and suggest unrated items of highest scores to the user, which is the same technique as that of *weighted-average* method. For selecting K nearest users, we use the *weight*, $w(a, i)$, that indicates how much the tastes of both user a and user i are similar. The predicted rating score of a user a for unrated item $j - K_{a,j}$ which is derived from Equation 1 – is represented as follows [1] [3]:

$$K_{a,j} = \bar{v}_a + k \sum_{i \in U_{a,K}} w(a, i)(v_{i,j} - \bar{v}_i) \quad [\text{Equation 5}]$$

Terms of Equation 5 are defined as follows:

$U_{a,K}$: a set of KNN of user a , and defined as follows:

$$U_{a,K} = \{i \mid \text{rank}(w(a, i)) \leq K\} \quad [\text{Definition 1}]$$

Terms of Definition 1 are defined as follows:

rank (): a function which ranks all the values in descending order

One of the concerns about this method is that although the system has a tremendous number of product items, active users tend to purchase only small percentages of the total items. The system can only consider evaluated items for recommendation. Therefore, the system based on *KNN* algorithms may be unable to make any good item recommendations for a specific user who has a unique tastes and therefore doesn't have common favorite items with other users [9].

2.1.4. Ratings

Rating values, $v_{i,j}$, are an indispensable resource of the memory-based methods – *weighted average* and *KNN* methods. Generally depending on how the recommender system is designed, rating values can be supplied by a user or generated by the software handling the data. How to collect the ratings data is also very important for making prediction.

There are three approaches to gathering ratings data in collaborative filtering:

- Active Filtering
- Passive Filtering
- Item-based Filtering

These three approaches are described below.

2.1.4.1. Active Filtering

Active filtering relies on explicit ratings provided by users. This filtering technique is used in a system where people with similar interests rate products and share the information with other people in the web. This is by far the best method of finding user opinions [20]. Here are examples of data collection in active filtering. The system asks users:

- to rate an item on a sliding scale;
- to rank a collection of items from favorite to least favorite;
- to choose the better of the two items;
- to create a list of items that they like;

In this project, we use the first one – to rate an item on a sliding scale – as active filtering method; this is the “user’s direct rating score on audio recording items” mentioned in Section 1.2.

One of the advantages of active filtering is an actual rating by a person who has already experienced the product; the recommender system can get rational information directly from a reliable source. Another benefit is that the recommender system can use these ratings without the need for preprocessing.

The disadvantages of the active filtering are:

- The system knows nothing about the item’s true nature or what its purpose is. The system only depends on preference ratings from users. Therefore it is highly possible that the users could get suggestion only about products with high rating values. Here are two possible situations:

- It is highly probable that when there are not many enrolled users in the system, the opinion could be biased by the existing users. In this case the systems could make wrong recommendations for new users based on the prejudiced data from existing users.
- If users decide not to participate in the rating work, the system could have insufficient data, and, therefore, the expectation results from system could be incorrect.
- New items or new users, before any rating is done on those items or by those users, are ambiguous to the system and may present issues such as the *first-rater problem* and the *cold-start problem* [20]:
 - The *first-rater problem* is caused by new items that have not received any ratings from users. The system can only recommend items which have rating scores. Therefore the system is unable to suggest these items until they get ratings.
 - Similarly, the *cold-start problem* is caused by new users who have not rated any item. Without any data about the user, it is impossible for the recommendation system to guess the user's tastes, predict what the user will like, and generate recommendations. Therefore, the new user cannot get any suggestions until she rates some items.

2.1.4.2. Passive Filtering

Passive filtering is the process of obtaining (computing) user ratings by observing each user's behavior on the system (web site). In this way the system does not need to solicit

user ratings, but rather, use user activity to determine their preferences. For example, a web server logs user visits to specific product item pages. Such information is used by passive filtering methods to compute an estimate of user ratings for the pages she visited.

Passive filtering systems rely on some of the following information [20]:

- User purchasing an item
- User showing interest by using, saving, and printing an item repeatedly
- User searching an item repeatedly
- User's social network and the like-minded users

Although the recommender system can get many kinds of observed data from user's behavior, those data are not an explicit user's rating score which can be direct input for a prediction calculation of user's preference. Therefore, for a passive filtering implementation, it is always required for the system designer to invent some rating measures to derive a user's rating score from the observed data. Detailed design of such measures is determined by the system designer and is hence application dependent.

An important feature of passive filtering is using the time aspect to detect the user's action. In some recommender systems, the time record helps to determine whether a user is scanning a requested web page or is fully reading the material on it [15]. In this project, we also used different time aspects to check how recently a user visited an item to speculate if the user is interested in the item.

The greatest strength of the passive filtering is that it doesn't require anything from users, and, at the same time, the systems can get a lot of data – usually much more data than active filtering – from every user automatically. However, as this method does not acquire the direct opinion of users, it cannot guarantee the accuracy of computed ratings.

Therefore we can assume that using passive filtering makes it harder to produce correct predictions and has a possibility of making a mistake when conjecturing a user's preference.

2.1.4.3. Item-based Filtering

Item-based filtering depends mainly on the relationships between items, not users, and anticipates which items would be recommendable products for a user by finding the items the user purchased; after the item is known, this method can make recommendations for the user by collecting items that are similar to the item the user purchased. This technique is widely used by Amazon.com, where the system recommends items to a user by showing the message, *“users who bought this item also bought the following items”*.

In detail, this method finds out how closely a target item and all other items are related to each other and makes the target item's list which contains items most closely related to the target item; the list becomes the target item's pre-made suggestion list. After that, anytime the system detects a user who has shown interest in a specific item, it suggests items in the specific item's pre-made list to the user.

Distinctive features of the item-based filtering are:

- Item-based filtering doesn't overload the system because the system uses a pre-made suggestion list and executes prediction calculation only when new items are added to the web service system; therefore the system usually doesn't spend time on correlation calculation and suggests recommendable items to a user promptly.

- The suggestion item list is static, not dynamic; in other words, usually the pre-made suggestion lists are not changed until new items appear and influence the existing correlations between items; such feature accounts for why the suggestion list of each item page, in the YouTube website, always shows the same recommendation items for users, as mentioned in Section 1.1.

2.1.4.4. Ratings Source For This Project

In this project, as user's direct rating scores on audio recording items – which provide the best information – were available, we used them for prediction. In addition, we collected and used rating values obtained via passive filtering techniques to gain more rating sources. We also gathered data which shows the correlations between items (audio recording titles) and used the data for an item-based filtering technique.

2.2. Related Work

Recommender systems appeared as a research topic in the mid-1990s when researchers began focusing on the recommendation problem that is related to the ratings structure [1]. In the same way, collaborative filtering methods have been one of the most renowned algorithms for the recommender systems and have been used for various products; for example, the Tapestry system for electronic documents [7], GroupLens for new articles [10], Amazon.com recommendation system for various items, and so on. In this thesis, we intended to primarily concentrate on the research about collaborative filtering for multimedia because our project involves a collection of audio recordings.

Here we discuss some trials of inventing an effective data filtering tool especially for multimedia. Baluja et al. suggested a new way of data filtering using view graphs for personalized video suggestions, and the technique, named Adsorption, is the methodology used in their study [2]. In collaborative filtering, accuracy of prediction is the key issue. To improve the accuracy, Tong Queue Lee et al. tried a time-based approach which depreciates the rating value according to the time lapse [11]. Yanchang Zhao et al. also proposed using a decaying function to deal with time series data [19]. In addition, Kazunari Sugiyama et al. tried different types of time-based collaborative filtering with detailed analysis of user's browsing history in one day [8].

There is a totally different approach to raising the accuracy of multimedia recommendation, especially for music. The existing music recommendation tools like that of iTunes know nothing about the nature of music files and they only rely on the rating values or the action of users. Depending only on such statistical data has limitations. In the existing music recommender systems, users can find music in only two ways. The first case is when the user knows the name of song or artist, and second case is when the system suggests music which got high rating scores from the other users who are like-minded. As a new attempt to overcome such limitation of existing music recommender systems, MusicSurfer came up with a content management system based on music descriptions which were retrieved directly from the music file. To find similar music, this system relies on the musical aspects like rhythm, tempo, harmony, melody and so on [4]. To get this data, the authors used techniques to analyze audio signals, identify the instrument, and figure out the mood. Another similar example is Pandora radio which was mentioned in Section 1.1. In Pandora's detailed recommendation

mechanism, over 400 different musical attributes are considered when selecting a song similar to a user's favorite song and these 400 attributes fall under several categories called as focus traits. It is known that there are around 2,000 focus traits. Examples of these are rhythm syncopation, key tonality, vocal harmonies and displayed instrumental proficiency [23]. Both MusicSurfer and Pandora require a large amount of information and analysis skills. However, in the near future, it is anticipated that such advanced and elaborate techniques will offer even higher accuracy in product recommendation and gain ascendancy in the recommender system area.

Chapter 3. Problem & Solution

3.1. The Problem

In the beginning this project had an affiliation with a start-up company whose business was to operate an on-line subscription service for audio recordings. The company had built an audio recording delivery web service system and the problem was they needed to equip a recommendation tool for users of their web service. In order to solve the problem we needed to understand the operations of the company, business model, customer interactions with software, and data available. Here we describe such details about the company's web site and the requirements for the recommender system.

3.1.1. Audio Recording Service Web Site

The company built a website for delivery of audio containing spoken word content. Service operation was designed to give users access to the recordings, allow them to select the ones they would want to peruse, and play the recordings back for the users.

3.1.1.1. Web Service System

The company has advertised the web site's unique *concise advice* format which is used to coalesce and distill advice from reputable *advisors* delivered in condensed and easy-to-digest *programs*. The web site's members become *users* of this content by way of one or more delivery channels for this audio content. The company offered a system of tiered membership subscription plans that afforded different degrees of access to the audio recordings: from limited number of recordings at a time to full unlimited access. The end

user could listen to recordings depending on the access limit specified by the plan.

Delivery channels included:

- A Flash-based player on the web
- A voice-recognition application accessed over the phone
- Download to Apple iTunes and iPod

User's activities on the web site are:

- Registration: Any web site visitor can register by creating an account.
- Login: User needs id and password to enter into the web service.
- Subscription Plan: Account holders can select a specific subscription plan.
- Subscription: User can listen to audio recording items according to the plan.
- Rating: User can rate any item which she subscribed

3.1.1.2. Database

Some of user activities listed above are regarded as user's behavior in the website and captured in the database that is used in conjunction with the system. The database evolves dynamically with users' behavioral and demographic changes, and also with introduction of new audio content. The database maintains all the information about users, titles, users' rating scores for titles, relation between title and segments, record for users' visiting on segments, relation between topics and titles, and so on. For data maintenance the system checks a user's behavior every time the user visits a segment and stores the time in the database. A regular update to this database does not run for more

than an hour and generally allows use of the web site's transactional database, either by segmentation of data or segmentation of server load onto a different server. In this way, all the users' behaviors are recorded in database system and the records are used as a resource for implicit data filtering.

3.1.2. Requirement to Recommender System

One of the requirements, stated by the company, to the recommender system was:

- The recommender system should aggregate information, calculate scoring related data, and filter patterns in an offline process that is intended to run nightly and maintain a database that can be used to service real time requests for recommendations.

When the web site was originally deployed, it ran without the recommender system. The idea was to collect some user behavior data to use in the development of the recommender system.

3.2. Solution

This section describes useful resources for the recommender system that we have developed.

3.2.1. Data Tables

All interaction data is stored in the database and used by the company's web site. Of the entire database, the following portions were of interest to us because they contained data

relevant to our recommender system. To protect the company's intellectual property, only data fields used for the recommender system are presented:

❖ In the tables below we employ the following abbreviations:

PK: primary key

INT: integer data type

DATETIME: date data type (YYYY-MON-DD format)

▪ `tbl_user_info` table

tbl_user_info		
PK	user_id	INT

[Table 1: `tbl_user_info`]

Information about users is collected in this table. (e.g. to detect how many users are enrolled in the web site). We only used the user ID to identify each user in the recommender system.

▪ `tbl_title` table

tbl_title		
PK	title_id	INT

[Table 2: `tbl_title`]

Information about the titles of the audio recordings is collected in this table. (e.g., to figure out how many audio title items the web site has). Again, only the ID of the recording title was used.

- `tbl_title_ratings` table

tbl_title_ratings		
PK	title_id	INT
PK	user_id	INT
	title_rating	INT

[Table 3: `tbl_title_ratings`]

Explicit user ratings are collected in this table.

- `tbl_title_segment` table

tbl_title_segment		
PK	segment_id	INT
	title_id	INT

[Table 4: `tbl_title_segment`]

Each title consists of multiple segments. `tbl_title_segment` table contains the list of segments for each title.

- `tbl_visited_segment` table

tbl_visited_segment		
	<code>title_id</code>	INT
	<code>segment_id</code>	INT
	<code>user_id</code>	INT
	<code>visited_date</code>	DATETIME

[Table 5: `tbl_visited_segment`]

When a user listens to audio recordings (segments), the database records all segments listened by the user and the dates when the user visited.

- `tbl_topic_title_map` table

tbl_topic_title_map		
	<code>title_id</code>	INT
	<code>topic_id</code>	INT

[Table 6: `tbl_topic_title_map`]

This table contains information of what title falls into what topic categories.

One title can be related to many categories.

In addition to data tables from the company's DB, we created our own data tables: `tbl_userbased`, `tbl_knearbased`, and `tbl_itembased`. We designed these three data tables as storage for each result from the three recommendation engines:

tbl_userbased		
PK	user_id	INT
	title_id01	INT
	title_id02	INT
	title_id03	INT
	title_id04	INT
	title_id05	INT
	title_id06	INT
	title_id07	INT
	title_id08	INT

[Table 7: tbl_userbased]

The tbl_userbased table stores prediction results from *weighted average method* engine.

tbl_knearbased		
PK	user_id	INT
	title_id01	INT
	title_id02	INT
	title_id03	INT
	title_id04	INT
	title_id05	INT
	title_id06	INT
	title_id07	INT
	title_id08	INT

[Table 8: tbl_knearbased]

The tbl_knearbased table stores prediction results from the calculation of *K-nearest neighbors* method engine.

tbl_itembased		
PK	user_id	INT
	title_id01	INT
	title_id02	INT
	title_id03	INT
	title_id04	INT
	title_id05	INT
	title_id06	INT
	title_id07	INT
	title_id08	INT

[Table 9: tbl_itembased]

The `tbl_itembased` table stores prediction results from the calculation of *item-based* method engine.

Each table contains 8 recommended audio recording titles for each user. `user_id` is the primary key which identifies each user. After these three tables are filled with results from each method, the web site software can use data in these three tables for outputting recommendations to users.

3.2.2. SQL Queries

To retrieve necessary data for our recommender system from the data tables described above, we used a variety of SQL queries. The queries are documented in Table 10.

Query	SQL	Query No.
To figure out how many users are enrolled the web site	SELECT MAX(user_id) FROM tbl_user_info;	[QUERY 1]
To figure out how many audio titles (items) the web site has	SELECT MAX(title_id) FROM tbl_title;	[QUERY 2]
To acquire rating values that each user rated on items	SELECT user_id, title_id, title_rating FROM tbl_title_ratings;	[QUERY 3]
To figure out how many segments are included in each title	SELECT title_id, COUNT(segment_id) FROM tbl_title_segment GROUP BY title_id;	[QUERY 4]
To figure out how many segments in a title were listened at least one time by each user	SELECT user_id, title_id, COUNT(DISTINCT segment_id) FROM tbl_visited_segment GROUP BY user_id, title_id;	[QUERY 5]
To figure out how many times segments in each title were listened by each user	SELECT user_id, title_id, COUNT(segment_id) FROM tbl_visited_segment GROUP BY user_id, title_id;	[QUERY 6]
To know when was each user's last day of visiting each title	SELECT user_id, title_id, MAX(visited_date) FROM tbl_visited_segment GROUP BY user_id, title_id;	[QUERY 7]
To figure out in which topic categories a title is included	SELECT title_id, topic_id FROM tbl_topic_title_map;	[QUERY 8]

[Table 10: SQL Query Table]

3.2.3. User Rating Data

The detailed methodology – how to transform *raw* data into *user rating* data type – is presented here. As mentioned in Section 1.2., to establish *user rating* data set, we import

two types of *raw* data: explicit user rating scores and data reflecting user listening patterns. In this project, the use of explicit ratings corresponds to the active filtering method and collecting implicit data related to user listening patterns falls under the passive filtering methodology.

3.2.3.1. Rating Data Source

We discuss from what data source we collected explicit and implicit data by using active and passive data filtering methods and how we designed the process of converting *raw* data to *user rating* data.

1) $A_{i,j}$: *user rating* from explicit user direct ratings

Stars	<i>raw data</i>	<i>user rating data</i> ($A_{i,j}$ or $B_{i,j}$)	Meaning
*	1	-4	I hate it.
**	2	0	I am Neutral.
***	3	2	It is OK.
****	4	4	I like it.
*****	5	5	I love it.

[Table 11: Rating Values]

To see how $A_{i,j}$ is acquired, on the web site, a user can rate on any audio recordings which she listened to. The rating score range is 1 ~ 5 stars. After the user rates a title with *a specific number* of stars, the rating value – *the specific number* – gets stored as a

raw datum (shown in Table 11) in the *tbl_title_ratings* table (shown in Table 3). When the recommender system needs user rating data, it retrieves the rating value from *tbl_title_ratings* table using SQL Query in QUERY 3 and converts the *raw data* to *user rating*, $A_{i,j}$, according to the conversion rule in Table 11. The meanings of rating values are also defined in the *Meaning* column. For example, if a user rates a title with three stars (***), it means the user thinks “the title is OK” and the three stars is changed to score number 3 (as *raw data*) and the number 3 is stored in *tbl_title_ratings* table. When the recommender system needs the score, it retrieves the number 3 and converts it to the number 2 (as *user rating*) and uses the number 2 for the prediction calculation.

2) $B_{i,j}$: *user rating* from implicit data related to listening patterns

To collect implicit data, the passive filtering approach is used. For this method, we designed three measures – *Weeks*, *Attachment*, and *Coverage* – and a condition table (Table 12). We need to decide the value of L_{score} – which denotes how much a *user i* likes a *title j* – by analyzing the implicit data which shows how often and recently the user visited the title. We model L_{score} of an audio recording based on three measures as follows:

- *Weeks* is number of weeks since the last date a user visited a title until now, in other words, it shows how many weeks a user has not visited the title. This measure is an important element which is related to time-based approach for L_{score} value decision. Therefore, the recommender system can deduce that a user’s more recent visit on a title reflects that the user is currently interested in the title and, as the value of *Weeks* increases, the preference of user becomes relatively

decreased [11]. Such a time-based approach is applied to recommender systems widely.

- *Attachment* represents how many times on average each recording segment in a title j was listened to by a user i . The computation of *Attachment* is defined as follows:

$$Attachment(i, j) = \min\left(10, \frac{AllListens(i, j)}{SegNumTitle(j)}\right) \quad [Equation 6]$$

$$(0 \leq Attachment \leq 10, i = \text{user id}, j = \text{item id})$$

Terms of Equation 6 are defined as follows:

$\min(x, y)$: this function selects smaller value between x and y

For example, when there are 10 segments (segment id 1~10) under a title j and if a user i listened to segment #1 10 times, segment #4 20 times, segment #8 7 times, and did not listen to all other segments, then the $Attachment(i, j)$ becomes $\min(10, (10 + 20 + 7) / 10) = \min(10, 3.7) = 3.7$. The maximum value of *Attachment* is limited to 10.

- *Coverage* is a percentage (%) which shows how many segments under a title j were listened to at least one time by a user i .

$$Coverage(i, j) = \frac{Listened(i, j)}{SegNumTitle(j)} \quad [Equation 7]$$

$$(0 \leq Coverage \leq 1, i = \text{user id}, j = \text{item id})$$

For example, if there are 10 segments under a title j and 6 out of 10 segments were listened by a user i , the $Coverage(i, j)$ becomes $6 / 10 = 0.6$. Therefore the maximum value of Coverage cannot exceed 1.

The following four variables are necessary to determine the values of *Weeks*, *Attachment* and *Coverage* values:

- *LastDate*: Last date a user visited a title
- *SegNumTitle*: Number of segments in a title
- *Listened*: The number of segments in a title which were listened to at least one time by a user
- *AllListens*: The number of times all the segments in a title were listened to by a user

By compound observations through the three measures, we could find some behavioral patterns of users and make relevant considerations as follows:

- i. If a user accessed any segment in a title most recently (in 1~3 weeks) and the user has not accessed other segments under the same title, we assume that the user didn't have enough time to listen to other segments yet and assume that the user likes the title.
- ii. If a user visited a segment in a title not frequently several months ago (in 4~12 weeks) and did not access many other segments under the same title, then we assume that the user is neutral on the title.

- iii. If a user visited segments in a title very frequently several months ago (in 4~12 weeks) and accessed many other segments under the same title, then we assume that the user loves the title.
- iv. If a user had visited segments under a title not very frequently many months ago (over 12 weeks) and did not access other segments under the same title, we assume that the user does not like the title.
- v. Although if a user visited many segments under a title very frequently many months ago (over 12 weeks) but did not visit the title recently, we assume that the user got interested in other titles but still likes the title.

Based on the user listening patterns described above, we designed Table 12. According to Table 12, the recommender system decides the L_{score} value with the given three measures' values and the L_{score} is regarded as *raw* data. Once L_{score} is decided, the L_{score} value (*raw* data) also should be converted to *user rating* data according to the conversion rule in Table 11. Finally the *user rating* data converted from L_{score} becomes the implicit rating data, $B_{i,j}$, which will be used for the prediction calculation.

Weeks	Attachment (A)	Coverage (C)	Means	L _{score} (Raw Data)
1 ~ 3	any value	any value	I like it.	****
4 ~ 12	$A \leq 0.5$	≤ 0.5	I'm Neutral.	**
4 ~ 12	$0.5 < A \leq 1$	any value	It's OK.	***
4 ~ 12	$1 < A \leq 2$	any value	I like it.	****
4 ~ 12	$A > 2$	any value	I love it.	*****
over 12	$A \leq 0.3$	≤ 0.3	I hate it.	*
over 12	$A \leq 1$	any value	I'm Neutral.	**
over 12	$1 < A \leq 2$	any value	It's OK.	***
over 12	$2 < A \leq 3$	any value	I like it.	****
over 12	$A > 3$	any value	I love it.	*****

[Table 12: Condition Table for L_{score}]

All the necessary variables for establishing *user rating* data, $A_{i,j}$ and $B_{i,j}$, can be obtained from the database by using SQL Queries as shown in Table 13:

Variables	Query No. (from Table 10)
<i>Number of users</i>	[QUERY 1]
<i>Number of titles</i>	[QUERY 2]
<i>SegNumTitle</i>	[QUERY 4]
<i>Listened</i>	[QUERY 5]
<i>AllListens</i>	[QUERY 6]
<i>LastDate</i>	[QUERY 7]

[Table 13: Variables and Relevant Queries]

3.2.3.2. Representative value $v_{i,j}$ from $A_{i,j}$ and $B_{i,j}$

After acquiring both $A_{i,j}$ and $B_{i,j}$ values, we need to choose one of them to construct the predicted ratings, $v_{i,j}$. To recall the notation used in this document:

- $v_{i,j}$: Representative *user rating* data chosen alternatively from $A_{i,j}$ and $B_{i,j}$
- $A_{i,j}$: *user rating* data from explicit user's direct rating
- $B_{i,j}$: *user rating* data from implicit data related to listening pattern

$v_{i,j}$ is decided by the following rules:

- a. When $A_{i,j}$ is not available and $B_{i,j}$ is available, use $B_{i,j}$.

$$v_{i,j} = B_{i,j}$$

- b. When $B_{i,j}$ is not available and $A_{i,j}$ is available, use $A_{i,j}$.

$$v_{i,j} = A_{i,j}$$

- c. When both $A_{i,j}$ and $B_{i,j}$ are available, choose the higher one.

- a) When $A_{i,j}$ is higher than $B_{i,j}$, $v_{i,j} = A_{i,j}$

- b) When $B_{i,j}$ is higher than $A_{i,j}$, $v_{i,j} = B_{i,j}$

- d. When neither $A_{i,j}$ nor $B_{i,j}$ are available, make the $v_{i,j}$ void so as not to be involved in the prediction calculation.

3.2.4. Sparse Matrix

In this project, for most of calculating works, we need to input all the available data into a matrix as shown in Figure 2 and execute the calculations – like correlations between users, correlations between titles, and prediction calculations – in the matrix environment. Therefore, we had to use a numerical application which was designed for matrix

calculation. We adopted the matrix-toolkits-java sparse matrix method (MTJ) which is a Java library that contains various matrix utilization functions and offers high calculation speed [13]. This tool was quite useful for implementation of our recommendation engines. In addition, we also utilized other matrices as temporary data storages when making suggestion lists. All the matrices used in this project were sparse matrices.

3.2.5. Recommendation Methods

Here we introduce basic methodologies for each of the three recommendation engines.

3.2.5.1. User-based Suggestion: *weighted-average* method

The *Weighted-average* method is one of the two user-based methods and we apply the *Memory-Based algorithm* to the prediction of the user's preference.

1) Equations for prediction

The anticipated rating score of active user a for title j , $P_{a,j}$ is calculated as follows:

$$P_{a,j} = \bar{v}_a + k \sum_{i=1}^n w(a,i)(v_{i,j} - \bar{v}_i) \quad [\text{Equation 1}]$$

The *weight* function, $w(a,i)$, in [Equation 1] is defined as follows:

$$w(a,i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \quad [\text{Equation 4}]$$

The *weight* is based on the rating scores of title j which both users a and i have rated.

Title j can be expressed as a member of the intersection set, $(I_a \cap I_i)$; where, I_a is a set of

titles which user a has rated and I_i is a set of titles which user i has rated. $(I_a \cap I_i)$ means a set of titles which both users a and i have rated.

2) Items (Titles) Suggestion

After finishing the calculations of prediction, the system can prioritize the titles according to their predicted rating scores. Then the system suggests the titles with the highest prediction scores to the user.

3.2.5.2. User-based suggestion: *K-nearest neighbors* method

The *KNN* method is almost the same as the *weighted-average* method. For example, let's say K is 20. Then the system selects 20 users to compare with a target user a . Those 20 users have the most similar preferences and tastes with user a . This method also uses the *Pearson Correlation Coefficient* to obtain the weight value. The number K can be decided by the system administrator, and, in this project, K is 20 because intuitively we thought 20 is proper for this project. Therefore the Equation 5 which was introduced in Section 2.1.3.3. is used for calculation of prediction, $K_{a,j}$.

$$K_{a,j} = \overline{v_a} + k \sum_{i \in U_{a,K}} w(a,i)(v_{i,j} - \overline{v_i}) \quad [\text{Equation 5}]$$

$U_{a,K}$ of Definition 1 can be applied as $U_{a,20} = \{i \mid \text{rank}(w(a,i)) \leq 20\}$

3.2.5.3. Item-based suggestion

In this method, for a user, the system suggests items (titles) which are closely related to the user's favorite item (title). In this project, to decide how closely titles are related, topic is chosen as correlation factor.

Topic Id Title Id							
	1	2	3	4	5	6	7
1	✓		✓		✓		
2		✓	✓		✓		
3			✓	✓		✓	
4	✓	✓	✓		✓		✓
5		✓		✓	..	..	..

* ✓: indicates what title is included in what topics' category

[Figure 3: Title-Topic Relation Matrix]

topic_id	topic_name
1	Attitude
2	Business
3	Cancer
4	Change
5	Child Development
6	Confidence
8	Death
9	Depression
10	...
...	...

[Table 14: tbl_title_topic_master]

The matrix of Figure 3 displays the relations between titles and topics based on the data from the *tbl_topic_title_map* table, Table 6, and every topic's id and name can be found in the *tbl_title_topic_master* table, Table 14.

To see how the Title-Topic Relation Matrix and Table 14 are used to find out relation between a title and topics, for example, we can see that title 1 falls into three topic categories (topic id 1, 3, and 5) in Figure 3 and the topic names (*Attitude*, *Cancer*, and *Child Development*) are given from Table 14. For calculation of the correlation between titles, we devised another weight, $w_I(a, i)$, which is defined in Equation 8:

$$w_I(a, i) = \frac{\text{Number of } (r_a \cap r_i)}{\text{Number of } (r_a \cup r_i)} \quad [\text{Equation 8}]$$

Terms of Equation 8 are defined as follows:

r_a : a set of all the topics that is related to title a .

The calculation of $w_I(a, i)$ is simple. For example, if we calculate $w_I(1, 2)$ in Figure 3, the calculation is as follows:

$$\begin{aligned} w_I(1, 2) &= \frac{\text{Number of } (r_1 \cap r_2)}{\text{Number of } (r_1 \cup r_2)} \quad (\text{ where } r_1 = \{1, 3, 5\} \text{ and } r_2 = \{2, 3, 5\}) \\ &= \frac{\text{Number of } (\{1,3,5\} \cap \{2,3,5\})}{\text{Number of } (\{1,3,5\} \cup \{2,3,5\})} = \frac{\text{Number of } (\{3,5\})}{\text{Number of } (\{1,2,3,5\})} = \frac{2}{4} = 0.5 \end{aligned}$$

In this way we can calculate correlations between other titles in Figure 3 as follows:

$$w_I(1, 4) = \frac{\text{Number of } (\{1,3,5\})}{\text{Number of } (\{1,2,3,5,7\})} = \frac{3}{5} = 0.6$$

$$w_I(2, 3) = \frac{\text{Number of } (\{3\})}{\text{Number of } (\{2,3,4,5,6\})} = \frac{1}{5} = 0.2$$

After calculations of correlations, $w_l(a, i)$, are completed, we can select titles which have the highest correlation values. Then the recommender system suggests the titles of the highest values to any user who liked or showed interest in title a .

Chapter 4. Implementation

4.1. System Architecture

The architecture of the recommender system is composed of several modular components, as illustrated in Figure 4. Both the *Direct Rating* (explicit rating data from user's direct rating) component and *Passive Rating* (implicit rating data from user's listening patterns) component retrieve *raw* data from the *Web Service Database* through the *DB Retrieve Connection* component. The system converts *raw* data into a representative *user rating* data source and then stores those *user rating* data into the *User Ratings* module. For the user-based methods, the *User Weight* module calculates *user weight* data – correlations between users. Using *user weight* data and *user rating* data from the *User Weight* module and *User Ratings* module, both the *Weighted Average Engine* & *K-Nearest Neighbors Engine* components make predictions. Then, based on the prediction results, the *Item Selection Function* component creates recommendation lists. For the item-based method, *item weight* data – correlations between items – are calculated and then stored in the *Item Weight* module. Using *item weight* data and *user rating* data from the *Item Weight* module and the *User Ratings* module, the *Item-based engine* component makes recommendation lists. All the recommendation lists from those three engines (*Weighted Average*, *K-Nearest Neighbors*, and *Item-based engines*) are moved to the *Database for Results* database via the *DB Store Connection* component and the web service system uses all the recommendation lists when it suggests items to users.

We implemented all the recommender system components in Java and employed MySQL DB as backend data storage for the *Web Service Database* and the *Database for Results*.

[Figure 4: Recommender System Architecture]

4.2. System Components

4.2.1. User-based Method

Figure 5 shows the user-based recommender system part out of the whole recommender system shown in Figure 4. The following is how the user-based recommender system works in detail:

- 1) The *Web Service Database* and the *Database for results* are databases, not components or modules.
- 2) The *DB Retrieve Connection* component uses JDBC for the connection between the *Web Service Database* and the *User-based Recommender System* software (inside of the dashed line), and retrieves necessary data from the *Web Service Database* by executing SQL Queries.
- 3) At first, the system procures all the necessary data from database. The data are:
 - Users' direct rating scores on titles
 - *LastDate*: the last date a user visited an title
 - *AllListens*: the number which shows how many times all the segments in a title were listened by a specific user
 - *Listened*: the number of segments (in a title) which were listened at least one time by a user
 - *SegNumTitle*: the number of segments contained in a title
- 4) In the *Direct Rating* component, the system converts all users' direct ratings into *user rating* type ($A_{i,j}$) according to Table 11.

[Figure 5: User-based Method Architecture]

5) In the *Passive Rating* component, the system decides the values of three measures – *Weeks*, *Attachment*, and *Coverage* – with variables as follows:

- *LastDate* is used for calculation of *Weeks*
- *AllListens* and *SegNumTitle* are used for calculation of *Attachment*
- *Listened* and *SegNumTitle* are used for calculation of *Coverage*.

With the decided values of three measures, the L_{score} value is determined according to the rules of Table 12 and then the L_{score} value is converted to *user rating* type ($B_{i,j}$) in accordance with the rules of Table 11.

6) In the *User Ratings* module, the system selects an adequate value from between $A_{i,j}$ and $B_{i,j}$ according to the rules explained in Section 3.2.3.2., and stores the selected value as the representative rating data, *user rating* ($v_{i,j}$).

7) In the *User Weight* module, with the representative rating data ($v_{i,j}$), the system calculates the *weight*, $w(a,i)$, between all the existing users using the function of Equation 4.

8) In the *Weighted Average Engine & K-Nearest Neighbors Engine* component, the system computes a prediction by manipulating both *user rating* ($v_{i,j}$) and *weight*, $w(a,i)$.

9) The *Weighted Average Engine* component uses the Equation 1 for prediction,

$$P_{a,j} = \bar{v}_a + k \sum_{i=1}^n w(a,i)(v_{i,j} - \bar{v}_i) \quad [\text{Equation 1}]$$

The calculation sequence of $P_{a,j}$ is:

- i. Select a target user a who will get recommendation.
- ii. Choose title j that user a didn't rate or show interest in.

- iii. Find all users i who gave a rating on item j .
- iv. Compute $P_{a,j}$ in Equation 1 as referring to data, $v_{i,j}$ and $w(a, i)$.
- v. For all the target users, repeat steps from i. to iv..

After finishing the calculations, the *Item Selection Function* component selects titles with the highest prediction values from the prediction result ($P_{a,j}$) and the selected titles become the elements of the recommendation list. Through the *DB Store Connection* component such recommendation lists for all users are stored in the data table, *tbl_userbased*, in the *Database for Results*.

- 10) In the *K-Nearest Neighbors Engine* component of this project, the number K can be decided by system administrator and we set K to 20. At first, for a user a , the system discovers 20 nearest neighbors based on the values of *weight* ($w(a, i)$). The set of 20 people can be presented using Definition 1 from Section 2.1.3.3.:

$$U_{a,K} = \{i \mid \text{rank}(w(a, i)) \leq K\} \quad (\text{where } K = 20) \quad [\text{Definition 1}]$$

Then the engine executes the function of Equation 5 for prediction,

$$K_{a,j} = \overline{v_a} + k \sum_{i \in U_{a,K}} w(a, i)(v_{i,j} - \overline{v_i}) \quad [\text{Equation 5}]$$

Calculation sequence of $K_{a,j}$ is almost the same as that of $P_{a,j}$. The difference is that the system refers to only 20 nearest neighbors' data for the prediction calculation as follows:

- i. Select a target user a who will get recommendation.
- ii. Choose item j that user a didn't rate or show interest in.
- iii. Find users i who rated on item j among the 20 nearest neighbors.
- iv. Calculate $K_{a,j}$ in Equation 1 as referring to data $v_{i,j}$ and $w(a, i)$.

- v. Repeat steps from i. to iv. for all the other target users.

After finishing the calculations, the *Item Selection Function* component selects titles with the highest prediction values from the prediction result ($K_{a,j}$) and the selected titles become the elements of recommendation list. Through the *DB Store Connection* component such recommendation lists for all users are stored in the data table, *tbl_knearbased*, in *Database for Results*.

4.2.2. Item-based Method

The item-based method is simpler than the user-based method. Figure 6 shows the item-based recommender system part out of Figure 4. The system works as follows:

- 1) The *Web Service Database* and *Database for results* are databases, not components or modules.
- 2) The *DB Retrieve Connection* component retrieves data which contains the relation between title and topic from the *Web Service Database*. Each audio recording title falls under multiple topics, and the relations between titles and topics are acquired from the *tbl_topic_title_map* table (Table 6) by executing SQL query, QUERY 8. After that, all the relations are used for calculation of the correlations between items.
- 3) In the *Item-Weight* module, the system calculates the corrections. The steps are as follows:
 - i. The relation between a title (row, represented i) and a topic (column, represented j) can be represented as $r_{i,j}$.

[Figure 6: Item-based Method Architecture]

- ii. If a title (row, represented i) is related to a topic (column, represented j), a number is given for the relation (e.g. $r_{i,j} = 5$); the number doesn't have special meaning and isn't used for calculation, but it is a flag which means the title i has relation with the topic j .
- iii. The system calculates the correlation, $w_I(a, i)$, between a specific title a and other title i . The correlation can be calculated by the function, Equation 8.

$$w_I(a, i) = \frac{\text{Number of } (r_a \cap r_i)}{\text{Number of } (r_a \cup r_i)} \quad [\text{Equation 8}]$$

(r_a is a set of all the existing relations between title a and any topic)

- 4) For a target title, the *item-based engine* component selects a *specific number* of titles that have the highest correlation values with the target title; the system administrator can determine the *specific number* and in this project we decided the *specific number* as 8 because we thought 8 is enough number of recommendation for each user. Therefore this system retains the each 8 nearest titles for all the titles; anytime when the system needs to suggest something to a user who liked a title, the title's 8 nearest titles are retrieved and used for the suggestion.
- 5) The *Item selection function* works in an additional process to make recommendation lists for users:
 - i. The *Item selection function* selects user's most favorite titles and sorts the titles from highest to lowest values.
 - ii. To choose recommendable titles, the *Item Selection Function* component looks at a user's favorite title, and then selects the favorite title's nearest titles. After that the recommender system suggests the nearest titles to the user.

- iii. Repeat step of ii. for all users.
- iv. Finally the system stores each list for each user into a database table in *Database for Results*. The table name is *tbl_itembased*.

4.2.3. Recommendation Lists in DB Tables

The web service system can use recommendation lists in data tables – *tbl_userbased*, *tbl_knearbased*, and *tbl_itembased* – anytime it needs to suggest items to users. Because the web service system was dead by the time we completed the recommender system, we were not able to integrate the recommender system into the company's software any further.

Chapter 5. Experimental Evaluation

This chapter describes the experimental tests to evaluate the three recommendation engines and shows the experimental results.

5.1. Comparative Analysis

This analysis is for comparing prediction accuracy of the three recommendation engines.

5.1.1. Terms

The following terms are used in the discussion that follows:

- *original start day*: the date the company's web service launched and started service
- *current date*: the word speaks for itself; current date, now, on which we're working on the analysis
- *pivotal date*: a date – which can be opted by us for analysis – located between *original start day* and *current data* can be chosen as *pivotal date*
- *prediction result*: every recommendation lists for every user created by each of the three recommendation engines based on the data created before *pivotal date*
- *actual behavioral history*: real record which contains every user's behavioral history and shows what titles every user has listened to during the period since the day after *pivotal date* until *current date*.

- *Sparsity*: the percentage of empty locations in the user-item ratings matrix. We can easily determine how sparsely the rating data are spread in the user-item matrix and it is defined as [16]:

$$Sparsity = 1 - \frac{Number\ of\ Ratings}{Number\ of\ Users \times Number\ of\ Titles} \quad [Definition\ 2]$$

5.1.2. Methodology

This test is comparing all *prediction results* based on the data created before *pivotal date* with *actual behavioral history* created after *pivotal date* and finding out how many correct predictions were made by the recommendation engines as follows:

- 1) We selected data accumulated during a specific period – since *original start day* until *Pivotal Date*.
- 2) Each of three engines makes a *prediction result* for all the users with the accumulated data and each *prediction result* is stored in the sparse matrices: *uBasedMTX* for the *prediction result* from the *weighted-average* method engine; *KnearBasedMTX* for the *prediction result* from the *K-nearest neighbors* method engine; and *iBasedRecMTX* for the *prediction result* from the *Item-based* method engine. Each matrix gets 8 recommendation titles for each user.
- 3) We also collected *actual behavioral history* data set from the database and stored in the sparse matrix, *ChangedMTX*.
- 4) After that, we checked if there are any identical titles between the titles contained in the three matrices – *uBasedMTX*, *KnearBasedMTX*, and *iBasedRecMTX* –

and titles in ChangedMTX, and counted the matching numbers for each of the three engines.

5.1.3. Results

We executed this test repeatedly by changing *pivotal date* as shown in Table 15.

Pivotal Date	Weighted Average	K-Near Neighbors	Item-based	Number of Users	Number of Ratings	Sparsity
2/1/2008	4	1	3	16	60	0.976415
2/8/2008	8	0	6	20	90	0.971698
2/15/2008	6	1	4	31	154	0.968756
2/22/2008	2	1	3	39	185	0.970166
3/1/2008	2	0	2	44	198	0.971698
3/8/2008	2	0	2	49	209	0.973174
3/15/2008	1	0	2	53	215	0.974487
3/22/2008	1	0	1	60	226	0.97631
4/1/2008	1	0	1	61	228	0.976492
4/8/2008	2	0	1	70	240	0.978437
4/15/2008	6	0	1	132	343	0.983657
4/22/2008	5	0	1	132	351	0.983276
5/1/2008	4	0	2	140	367	0.983513
5/8/2008	4	0	2	147	377	0.98387
5/15/2008	3	0	2	152	383	0.984153
5/22/2008	4	0	2	157	389	0.984417
6/1/2008	3	0	2	165	402	0.984677
6/8/2008	2	0	2	175	424	0.984762
6/15/2008	3	0	2	176	430	0.984634
6/22/2008	3	0	2	178	438	0.984524
7/1/2008	4	0	0	189	461	0.984659
Total	70	3	43			

[Table 15: Comparative Analysis Result Table]

In Table 15,

- *Number of Users* means the number of active users who rated titles (explicit data) or whose listening patterns caused the generation of rating values (implicit data).
- *Number of Ratings* is the total number of explicit and implicit ratings caused by all the active users.
- As the *pivotal date* increases, the values of both *Number of Users* and *Number of Ratings* are increased naturally.
- The three columns, named *weighted-average*, *K-nearest neighbors*, and *item-based*, show how many times each engine made correct predictions by comparing the each *prediction result* with the *actual behavioral history* of users.
- In total, the *weighted-average* method made 70 correct predictions with its prediction, the *K-nearest neighbors* method made only 3 correct predictions, and the *item-based* method made 43 correct predictions. The *weighted-average* engine shows to be most effective among the three recommendation engines in this experiment.
- Although the *K-nearest neighbors* method is also based on collaborative filtering and pretty much similar to the *weighted-average* method in mechanism, this method was not effective because it couldn't produce enough *prediction result* data. To explain the reason, when we wanted to predict a user a 's preference for title j , we could hardly find out other user who rated title j among user a 's 20 nearest neighbors. Without relevant users who rated the same title j , we were unable to predict user a 's preference to title j ; and such condition restrained the *K-*

nearest neighbors method from producing enough *prediction result* and thereby the efficiency went down.

- The value of *sparsity* is changed according to some factors. For example, if the *Number of Ratings* is increased, the *sparsity* is decreased; on the other hands, if the *Number of Users* or *Number of Titles* are increased, the *sparsity* is increased. Therefore as the *Pivotal Date* is changed, such factors – *Number of Ratings*, *Number of Users*, and *Number of Titles* – are also changed and consequently the value of *sparsity* is also changed. In our experiment, as the *Pivotal Date* has been increased, the *sparsity* has been increased because as time goes by users' activity has been increased.

5.2. Accuracy Evaluation

This test is for accuracy evaluation of the *weighted-average* method. Our primary concern has been the *weighted-average* method and, as we expected in Section 1.2., *weighted-average* method, which is a kind of user-base filtering method, showed the best performance with audio recordings among the three engines as shown in Section 5.1. Given this result, we tried one more test. We applied an *empirical analysis approach* – which is widely known as a proper testing method for collaborative filtering algorithm – to our accuracy evaluation design [3].

5.2.1. Data Set

In this test, the data set contains rating information of 1814 users on 159 audio recording titles and has 1051 rating scores. All the data are stored in a user-item sparse matrix whose each row number indicates each user and each column number indicates each recording title like the matrix in Figure 2. The *sparsity* of the entire data set is calculated as:

$$1 - \frac{\text{Number of Ratings}}{\text{Number of Users} \times \text{Number of Titles}} = 1 - \frac{1051}{1814 \times 159} = 0.9964.$$

We divided the existing *user rating* data set into a *training set* and *test set* randomly [3] and in this evaluation we mainly focused on the *test set*. Every time we executed tests, we increased the number of *test set* from 1 to 105 out of total 1051 ratings.

Here are explanations of some terms:

- P_a : *test set*
- a : user id, where user a is a member of the *test set*
- S : the sum of absolute deviation of all user in the *test set*
- A : average absolute deviation for a user
- $p_{a,j}$: predicted rating score of user a for title j , where user a is in the *test set*
- $v_{a,j}$: actual rating score of user a for title j , where user a is in the *test set*
- m : the number of the *test set*

The S and A are defined and calculated for the *empirical analysis approach* as follows [3]:

$$S = \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \quad [\text{Equation 9}]$$

$$A = \frac{1}{m} \sum_{j \in P_a} |p_{a,j} - v_{a,j}| \quad [\text{Equation 10}]$$

To evaluate how precisely the *weighted-average* method makes its prediction, we calculated the A value – the average absolute deviation value – and observed the result.

5.2.2. Methodology

Original rating score values are -4 (I hate it), 0 (I am Neutral), 2 (It is OK), 4 (I like it), and 5 (I love it) as shown in Table 11. Therefore the rating score range is 9 (-4 ~ 5).

The detailed procedure for the test is:

- 1) We know m existing rating scores ($v_{a,j}$) of the *test set*.
- 2) As input process of this test, we converted the m existing rating scores ($v_{a,j}$) to unrated (null) status. m is the number the *test set*.
- 3) After the prediction calculation, prediction values ($p_{a,j}$) for the unrated status (null) scores are created as output.
- 4) After that, with $v_{a,j}$, $p_{a,j}$, and m values, we calculated S in Equation 9 and A in Equation 10. The result of these two equations for S and A is shown in Table 16.

5.2.3. Results

Results of the evaluation test are shown in Table 16, and Figure 7 is created from the results of the Table 16. In the ratio of (A / Rating Score Range), the Rating Score Range value is 9 (-4 ~ 5).

Number of users for test (m)	Sum of Deviations (S)	Average Deviation (A)	Average Deviation Ratio (A / Rating Score Range)
1	4.67	4.67	51.9%
2	9.03	4.51	50.1%
3	9.03	3.01	33.4%
4	9.76	2.44	27.1%
5	13.76	2.75	30.6%
9	34.37	3.82	42.4%
16	65.32	4.08	45.4%
22	69.42	3.16	35.1%
30	81.56	2.72	30.2%
33	88.96	2.70	30.0%
35	90.66	2.59	28.8%
39	90.66	2.32	25.8%
42	94.63	2.25	25.0%
46	112.94	2.46	27.3%
49	120.52	2.46	27.3%
52	129.52	2.49	27.7%
56	134.41	2.40	26.7%
60	146.03	2.43	27.0%
66	173.92	2.64	29.3%
69	192.62	2.79	31.0%
70	218.70	3.12	34.7%
74	227.76	3.08	34.2%
80	235.76	2.95	32.7%
85	248.92	2.93	32.5%
90	258.30	2.87	31.9%
95	287.61	3.03	33.6%
99	298.81	3.02	33.5%
105	317.51	3.02	33.6%

[Table 16: Accuracy Evaluation Result Table]

As the number of users for test, m , increases, the sum of the absolute deviation, S , increases, and the average absolute deviation for a user, A , comes close to 3 (33%, Average Deviation ratio).

In Figure 7, the *Sum of Deviations* (S) shows how the whole errors are increased as the number of users of test set (m) increases. The *Average Deviation* (A) is a very important factor for estimating how the weighted-average method can work with constant accuracy.

[Figure 7: Chart for Deviation Result]

With the graphic result we could confirm that the *weighted-average* method, based on collaborative-filtering method, has a highly accurate aspect for data prediction.

Chapter 6. Conclusion / Future Work

6.1. Achievement & Conclusion

Recommender systems are great tools for expanding the value of web service businesses and have become representative trendy technology in the web service industry. A lot of research and development is focusing on this topic. This document described the implementation of a recommender system for audio recordings. Although we adopted some concepts, theoretical functions, and methodologies, we designed our own system architecture. We also devised testing methods, compared the three different recommendation engines, and evaluated the accuracy of the *weighted-average* method. In the process of testing and analysis, the *weighted-average* method outperformed other methodologies. Therefore we conclude that the *weighted-average* method is the best fit for audio recordings recommendation in our application domain.

6.2. Future Work

Although the *weighted-average* method is the best for multimedia, we assume that properly combining the *weighted-average* method and the *item-based* method could be better and more effective than using only the *weighted-average* method because, for items like movie, we intuitively perceive that the *item-based* method is better than other methods as discussed in Section 1.1. Therefore studying how to assign the contributions of both *weighted-average* method and *item-based* method according to the character of

recommendation item and comparing the combined method and single *weighted-average* method would be a very interesting area of further study.

In the test environment, we had over 1800 users and over 150 items. For calculation of the prediction for each user or the correlations between items, it takes time. In this project, the time complexity for correlations between users (or between items) is $O(n^2)$. If we assume that there are multi-million users and multi-hundred thousand items, the prediction calculation time will be increased substantially. For this scalability problem, research on the solution to decrease the time complexity would be one of the most important issues in the user-based collaborative filtering area.

REFERENCES

- [1] Adomavicius, G., Tuzhilin, A., 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. on Knowl. and Data Eng.* 17, 6 (Jun. 2005), 734-749. DOI= <http://dx.doi.org/10.1109/TKDE.2005.99>
- [2] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. 2008. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceeding of the 17th international Conference on World Wide Web* (Beijing, China, April 21 - 25, 2008). WWW '08. ACM, New York, NY, 895-904. DOI= <http://doi.acm.org/10.1145/1367497.1367618>
- [3] John S. Breese, David Heckerman, and Carl Kadie. 1998. (July), Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*.
- [4] Cano, P., Koppenberger, M., and Wack, N. 2005. An industrial-strength content-based music recommendation system. In *Proceedings of the 28th Annual international ACM SIGIR Conference on Research and Development in information Retrieval* (Salvador, Brazil, August 15 - 19, 2005). SIGIR '05. ACM, New York, NY, 673-673. DOI= <http://doi.acm.org/10.1145/1076034.1076185>
- [5] Cöster, R. and Svensson, M. 2005. Incremental collaborative filtering for mobile devices. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (Santa Fe, New Mexico, March 13 - 17, 2005). L. M. Liebrock, Ed. SAC '05. ACM, New York, NY, 1102-1106. DOI= <http://doi.acm.org/10.1145/1066677.1066926>
- [6] Gil, A. and García, F. 2003. E-commerce recommenders: powerful tools for E-business. *Crossroads* 10, 2 (Dec. 2003), 6-6. DOI=<http://doi.acm.org/10.1145/1027328.1027334>
- [7] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12 (Dec. 1992), 61-70. DOI= <http://doi.acm.org/10.1145/138859.138867>
- [8] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th international conference on World Wide Web*, pages 675 - 684, New York, NY, USA, 2004.

- [9] Kim, D., Atluri, V., Bieber, M., Adam, N., and Yesha, Y. 2004. A clickstream-based collaborative filtering personalization model: towards a better performance. In *Proceedings of the 6th Annual ACM international Workshop on Web information and Data Management* (Washington DC, USA, November 12 - 13, 2004). WIDM '04. ACM, New York, NY, 88-95. DOI= <http://doi.acm.org/10.1145/1031453.1031470>
- [10] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. 1997. GroupLens: applying collaborative filtering to Usenet news. *Commun. ACM* 40, 3 (Mar. 1997), 77-87. DOI=<http://doi.acm.org/10.1145/245108.245126>
- [11] Lee, T. Q., Park, Y., and Park, Y. 2008. A time-based approach to effective recommender systems using implicit feedback. *Expert Syst. Appl.* 34, 4 (May. 2008), 3055-3062. DOI= <http://dx.doi.org/10.1016/j.eswa.2007.06.031>
- [12] Linden, G., Smith, B., York, J. 2003, Amazon.com recommendations: item-to-item collaborative filtering., *Internet Computing*, IEEE, Volume 7, Issue 1, Jan/Feb 2003 Page(s): 76 - 80., Digital Object Identifier 10.1109/MIC.2003.1167344
- [13] MTJ (matrix-toolkits-java), <http://code.google.com/p/matrix-toolkits-java/>
- [14] NetLingo, scrobble, <http://www.netlingo.com/word/scrobble.php>
- [15] Parsons, J., Ralph, P., and Gallagher, K. 2004. (July), Using viewing time to infer user preference in recommender systems., *AAAI Workshop in Semantic Web Personalization*, San Jose, California
- [16] Sarwar, B., Karypis, G., Konstan, J., and Reidl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international Conference on World Wide Web* (Hong Kong, Hong Kong, May 01 - 05, 2001). WWW '01. ACM, New York, NY, 285-295. DOI= <http://doi.acm.org/10.1145/371920.372071>
- [17] Sotomayor, R., Carthy, J., and Dunnion, J. 2005. The Design an Implementation of an Intelligent Online Recommender System., *American Association for Artificial Intelligence. Technical Report FS-05-07* Published by The AAAI Press, Menlo Park, California.
- [18] Weigend, A. S. 2003. Analyzing customer behavior at Amazon.com. In *Proceedings of the Ninth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining* (Washington, D.C., August 24 - 27, 2003). KDD '03. ACM, New York, NY, 5-5. DOI= <http://doi.acm.org/10.1145/956750.956754>

[19] Y. Zhao, C. Zhang, and S. Zhang. A recent-biased dimension reduction technique for time series data. In Advances in Knowledge Discovery and Data Mining: 9th Pacific-Asia Conference, PAKDD 2005, volume 3518 / 2005. Lecture Notes in Computer Science, 2005.

[20] Wikipedia, Collaborative Filtering,
http://en.wikipedia.org/wiki/Collaborative_filtering

[21] Wikipedia, Information Filtering,
http://en.wikipedia.org/wiki/Information_filtering_system

[22] Wikipedia, Last.fm, <http://en.wikipedia.org/wiki/Last.fm>

[23] Wikipedia, Pandora (Music Service),
[http://en.wikipedia.org/wiki/Pandora_\(music_service\)](http://en.wikipedia.org/wiki/Pandora_(music_service))

[24] Wikipedia, Recommender System,
http://en.wikipedia.org/wiki/Recommender_system