A Distributed Multi Event Solution for Recommender Systems Using Hadoop

Seyed Javad Seyedzadeh Kharazi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
September 2018
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Adnan Acan
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Adnan Acan
2. Assoc. Prof. Dr. Mehmet Bodur
3. Asst. Prof. Dr. Mehtap Köse Ulukök

ABSTRACT

Big data is a phenomenon that takes central stage in industry and academia arising from the advent of online services and mobile applications. Improving the efficiency of data processing and analysis has become a challenging issue. While a number of methods from different communities have been proposed for solving the "Big Data" problems, we worked with multi-event Intelligent Systems that offer efficient mechanisms, which significantly reduce the costs of processing large volume of data and improve data processing quality. Social networks could benefit from recommender systems in order to optimize the queries and ads they display for each special user. Among different approaches to analyze user data and making recommendations, we employed Collaborative Filtering with Cosine Similarity criterion for item-based similarity recognitions. In the implemented method, a Holonic multi-event system (HMES) is designed to process a portion of Amazon database in a distributed manner. The use of Hadoop and map-reduce technology is aimed to make more accurate and faster predictions and recommendations. Different evaluation standards such as Perfect Hit (PHIT), and Mean Percentage Rank (MPR) are used to examine and compare the proposed method with other conventional methods. The results obtained in this thesis are satisfactory compared to the results of the evaluation given in the literature.

Keywords: recommender system, hadoop, multi event, artificial intelligence, big data, holonic

ÖZ

Büyük veri, çevrimiçi hizmetlerin ve mobil uygulamaların ortaya çıkmasından kaynaklanan endüstri ve akademi merkezini alan bir olgudur. Veri işleme ve analiz verimliliğinin artırılması zorlu bir konu haline gelmiştir. “Büyük Veri” sorunlarının çözümü için farklı topluluklardan bir takım yöntemler önerilmişken, çok sayıda verinin akıllıca işlenmesi ve veri işleme kalitesini iyileştirme maliyetlerini önemli ölçüde azaltan etkili mekanizmalar sunan Çok-olaylı Akıllı Sistemler ile çalıştık. Sosyal ağlar, her bir özel kullanıcı için görüntüledikleri sorguları ve reklamları optimize etmek amacıyla öneri sistemlerinden yararlanabilir. Kullanıcı verilerini analiz etmek ve önerilerde bulunmak için farklı yaklaşımlardan ötürü, maddeye dayalı benzerlik tanımları için İşbirlikçi Filtrelemeyi Kosine Benzerlik kriteri ile birlikte kullandık. Önerilen yöntemde, bir Holonik Çok-olaylı sistem (HMES), Amazon veritabanının bir kısmını dağıtılmış bir şekilde tasarlamaktadır. Hadoop ve harita azaltma teknolojisinin kullanılması daha doğru, daha hızlı tahminler ve önerilerde bulunmayı amaçlamaktadır. Önerilen metodu diğer geleneksel yöntemlerle incelemek ve karşılaştırmak için, Mükemmel Vuruş (PHIT) ve Ortalama Yüzde Oranı (MPR) gibi farklı değerlendirme kriterleri kullanılmaktadır. Literatürde verilen değerlendirme sonuçlarıyla karşılaştırıldığında bu tezde elde edilen sonuçlar memnuniyet vericidir.

Anahtar Kelimeler: Öneri Sistemi, Hadoop, Çok Olaylı, Yapay Zeka, Büyük Veri, Holonik

I would like to dedicate this thesis to my family – to my beloved parents Mr. Reza Seyedzadeh and Mrs. Haideh Yousefi for their endless support, to my loving sisters Negin and Narges for keeping my spirit up with all the innocence and their never-ending motivations, and to my brother Jalal for his constant encouragement to accomplish the thesis work. Last but not least, this thesis is dedicated to my special friend Selin Tansu Tunç who has accompanied me through every effort and thought of this thesis.

ACKNOWLEDGMENT

First and separated of all, I would like to express my deepest appreciation to my advisor, Assoc. Prof. Dr. Adnan Acan for giving me an opportunity to work with and for his advice, encouragement and constant support. I would like to thank him also for his invaluable feedback and comments throughout the course of this project and on the thesis. He always gave me his time, even on his vacations. He has been a source of motivation, and I thank him for his gracious and benevolent support.

I want to thank Prof. Dr. Işık Aybay, Assoc. Prof. Dr. Önsen Toygar and Asst. Prof. Dr. Ahmet Ünveren who plant the first seeds of this work throughout the courses and personal guidances, and also the rest of the faculty members of Computer Engineering department and the staff for all their hard work and dedication. They made my study at the Eastern Mediterranean University, a pleasant and memorable one.

I would like to thank my thesis committee members Assoc. Prof. Dr. Mehmet Bodur and Asst. Prof. Dr. Mehtap Köse Ulukök for reviewing my thesis.

I would like to thank Armin Mehri for his valuable inputs on debugging the system. I would also thank Amin Hosseini Marani for timely code reviews and his feedback on result distribution. I would also thank Selin Tansu Tunç for her advices on the literature of the thesis and her efforts on the linguistic competence of the work.

And finally, I must thank my family for supporting me during the development of this work with no complaints and continuing, loving support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Foreword

Expansion of data storage and processing technologies has made life much easier in aspect of data management and analysis. It is now possible to store and retrieve huge amounts of data in less time than ever before by benefiting of the modern methods. It is hard to find a company, organization or even a small shop that is not willing to use computer in order to categorize and manage its information. Improvement of data storage capabilities and hardware abilities both got together and formed the concept of big data everywhere around us. Data mining is devised to help us analyze these big data and interpret and attain a specific direction of the raw data.

Online shops, polling websites and social networks are facing the presence of a huge number of users and have become a huge base of raw data of users with access because of storing all the relative information. Whether if the sites are designed for sales or polling or even entertainment purposes, they will use their data in order to improve the quality of the site and provide their services to increase audiences, and sooner or later they should give up their places to those competitors whom are putting more effort on data process. If online stores know what their customers intend to buy along with other products, they will definitely boost sales. It is enough to offer the item Y to the customer at the time of purchasing the item X, provided that the store knows as a fact that the customer who bought item X was also interested in item y. Of course, the customer will also benefit of this and will be able to make purchases more easily with offers from the store. In social networks, data analysis can also be beneficial to the site operators. If it is known which content the user likes more, the operators should expect the user to spend more time on their social platform by managing to show them similar contents of their interests. This can also be used to benefit of displaying relative ads on the side of the platform. It is obvious that displaying well-received ads are more welcomed than annoying the audience by irrelevant advertisements that they are not interested in.

Recommender systems are defined as a technology that is extended in the environment where items are to be recommended to users, or the opposite. These systems help users, customers, or readers to find a content, product, or article of their own interest. Naturally, these systems will not be able to offer without proper and correct information about users and their items (such as movies, music, and books). A custom-built RS is a must to recommend the most valuable information to the customers of an online store (Wu, Zhang, & Lu, 2015). Hence, one of their most basic goals is to collect information according to the users' preferences and also the existing items in the system. There are various sources and methods for collecting such information. An approach is explicit data collection, in which the user explicitly announces what he likes (for example, by rating a song or placing 5 stars for a movie). The other method is implicit information, which is a bit more difficult to collect. In this case, the system must record the user's tastes by controlling and following its behaviors and activities (for example, what the user is listening to or what content they are watching or who they are associated with, should be observed). In addition to the implicit and explicit information, some systems use users' personal information. For example, age, gender, and nationality of users can be a good source for cognition of the user. This kind of information is called demographic data, which a group of recommender systems is based on (Resnick & Varian, 1997).

In the recent years, many researches have been conducted in the area of data collection, and several articles have been proposed and published. Sometimes the queries that a user makes on the system, or the expression of their request details, will guide the system not only to identify the user's item, but also will help the system identify new information such as volume, power consumption and even custom colors of the user's items, and use them in future recommendations. In general, user query records are a huge and useful resource for tracking the user's tastes and interests. Having this information stored and processed, the result can be used to improve the efficiency of future suggestions offered by the recommender system.

The systems that behave based on queries are known as knowledge-based recommender systems. Another type of recommender system is the content-based model. In these systems, users' behavioral similarity will only be based on their writings and comments and on the keywords of the texts. Each text will be evaluated and categorized based on the keywords in that text. In addition to that, the keywords of a text based on the characteristics of an item (which has a user's opinion recorded for it), expresses the degree of satisfaction or discontent of the user with respect to the specified attributes. In content-based systems, the system, by extracting keywords that are repetitive words in the texts and essentially define the semantic direction of the user, determines and addresses the degree of satisfaction or dissatisfaction of the user and adjusts its recommendations based on the user's writings.

1.2 Problem Statement

One of the methods for analyzing user information in recommender systems is the use of neighboring and similarity calculation methods. This method uses previous user behaviors and analyzes the relationships between users and the dependencies of those behaviors on products (such as selecting an item, or clicking on a link, etc.) to identify a user and item of interest to him or her. In fact, whatever is stored in this system is the history of the behavior of the user in the face of products, services or comments and hence, this recorded information is important. One of the most successful neighboring models is the cosine similarity calculation method used in recent years. In the cosine similarity method, based on the data in the database (online store, social network, etc.), a user-item matrix is formed, which includes the user's rating to an item or selection of the item by the user. This matrix is also called the rank matrix (Parambath, 2013).

One of the two main problems in calculating the cosine similarity is the speed of this model, because for each user, all other users need to be analyzed so that common products can be introduced to users. This reduction of speed may not be very tangible for a thousand users, but when it is about a few million users of a huge platform such as Amazon, it will certainly not be ignored. Speed reduction is not just dependent on the number of users and the number of items is also very influential. A couple of tens of millions of products on Amazon should be reviewed for each pair of users, or at least a large subset of them will be analyzed, which will definitely impose a lot of computing load to any system. Using Hadoop technology to perform distributed calculations is an ideal idea for such a high-volume recommender system that deals more with live streams of data.

Using Hadoop alone will not solve the problem of reviewing a huge amount of data from a massive database such as Amazon. There are no proper facilities such as cluster servers available on the scale of laboratory, and it should be possible to run the model on even simpler systems with lower performances. The map-reduce technology in the Hadoop is the perfect solution to fix this problem. Using map-reduce will only allow the executable part of the database to enter the RAM and only the same section will be processed. Of course, how to combine the output of each run up of Hadoop is a challenge that is discussed along with the use of averaging in the third chapter.

The second problem with the cosine similarity of the recommender system is the need to train the entire train set at once, while it is impossible for large website servers that record thousands of new comments every second. If the whole train set has to be trained every time, it may be necessary to repeat this operation every day or every hour. In the proposed method, a Holonic system is designed using a multi-agent definition. Each agent has a separate task and can operate in parallel with other agents. The other feature of the Holonic multi-event System (HMES) is the ability to use multiple operating agents per run and even using multiple agents of the same type, thus expecting the speed to increase impressively without decreasing accuracy.

1.3 Research Hypothesis and Questions

The research ahead is based on the hypotheses that will be presented briefly below. These assumptions are considered in implementing the proposed model and comparing similar methods and finally the simulations:

- The basic hypothesis is that the information of each user, the items, and their ratings, plus the time of the recorded comment, is the accessible data.
- The score for each item is between 1 and 5, and the value of 0 for each element of the rating matrix means the absence of the user i's response to the item j.

- The matrix elements update simultaneously, and the error will be recalculated after the changes are made.
- Simulation of all models will take place on the same computer.

Along with the hypothesis, before starting the research and starting the simulations, fundamental questions will be raised about the existing methods and the implemented method that will be answered directly or implicitly during the research.

1. Does using a HMES method cause speed enhancement?
2. How will a HMES method change the error rate?
3. What is the difference in runtime between the implemented model and similar models?
4. Will implementation of map-reduce and combining of the results have a positive or negative effect on the final results?
5. Which of the suggested or compared methods has a better performance?

1.4 Research Objectives

The main purpose of the research is to investigate the proposed method in comparison with mathematical models, neighborhood similarity and matrix factorization such as gradient descent in terms of accuracy and speed in finding the least error. During the research, a model will be proposed, which, in addition to having the characteristics of the HMES algorithm, has the ability to run parallel and also increase the accuracy and speed, not only with respect to the simple models of the cosine similarity algorithm, but also has advantages than the other regular models such as the descending gradient. The other goal of this research is as follows:

- Increasing the execution speed of the entire program
- Increasing the accuracy of the recommender system
- Review the similar calculation methods and find the best model

1.5 Thesis Structure

Having the introduction to the thesis conducted, we will continue with the rest of the work by explaining the structure of the thesis. In the second chapter, literature review of the research and the principles of the work will be elaborated in detail. In this chapter, we will examine the general methods of recommender systems in detail and explore several commonly used methods by giving the definitions and examples. In the third chapter of the study, methodology of the proposed method combining of cosine similarity calculation, the map-reduce library of the Hadoop framework and the Holonic multi-event system will be explained. The fourth chapter will contain a review of the performed simulations and an analysis of the conducted experiments. At the end of this research, chapter five is aimed to provide the suggested future work on the subject and an overall conclusion of the study.

Chapter 2

LITERATURE REVIEW

2.1 Recommender Systems

The Recommender System, based on the collection of user behavioral information, can make suggestions, such as what music to listen or what to read, and even what goods to choose and buy from the sales site, by providing data mining techniques. A desirable RS is a system that using the dynamic and state-of-the-art data processing methods provides semantic data, which can be personified for different users (Aggarwal, 2016).

Almost any Internet user is somehow familiar with Recommender Systems, and they have worked with at least one of the existing types. Websites such as amazon.com and many shopping sites, review sites, and critique films like MovieLens are recommender systems that filter and share information with intelligent methods by collecting and retrieving user opinions. This is a Collaborative Filtering (Adomavicius & Tuzhilin, 2005). Figure 2-1 shows what is happening behind the scenes of a Recommender System in simple language.

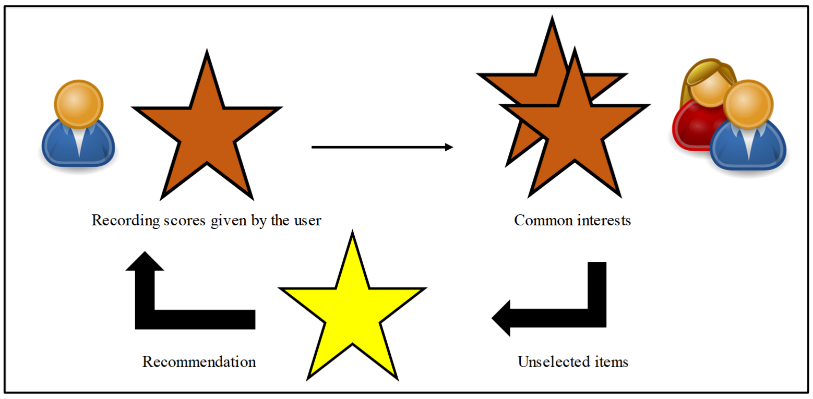


Figure 2.1: Showing the behavior of the Recommender System

In a nutshell, the work of the system can be described in such a way that a user will be considered as a goal for the system by entering the site. The Recommender system will offer suggestions to increase productivity (increase sales, item selection, content display, or user satisfaction) based on user's interests and other users who are similar to that particular user. These suggestions are based on other users' preferences. The preferences are that when user x and other users choose the item y and if the other users prefer the item z, then the item z will be a candidate item to be introduced to x. Style information and user choices are all stored in the database until they are refined and processed at a time when people with the same interest are found. The main features of a recommender system can be summarized as follows:

2.1.1 User-Based Information

The most important feature of an RS is to collect data based on user behavior on the site and the interests of users who are registered on the site. The significance of this feature becomes bold when it comes to knowing that some systems should make a recommendation according to a simulation of user behavior and only based on estimations.

2.1.2 A simpler Search

If what the user's orientation and interest in choosing items is known, proposing to them will not only help the system but will also help the user to search in a space that has never before had the chance to search that space or was not easily accessible to him.

2.1.3 Affecting by Similar Users' Features

Users who have behaved similarly to a user's behavior in choosing items can better guide system to recommend him. Those who share an interest in choosing items like computer games may have other common interests. While an extensive variety of data is available thanks to the growth of social media and e-commerce, the science of BigData logical analysis could derive a benefit of the existence of modern data architectures of non-relational data to grow even bigger (Venkatraman, Fajd, Kaspi, & Venkatraman, 2016).

2.1.4 Up-to-date Information

The Recommender system, based on a database of all users and items, can provide suggestions in line with the interests of users. For a more productive system of proposers, the database of these systems should always be up to date, because as a matter of fact, the interests and choices of users are changing rapidly and differently.

2.1.5 Reduce Costs

The information in the database of a recommender system, without any cost and only with the help of users, will be recorded in the system, and based on different methods of data analysis and processing a new offer is presented. No cost will be charged to extract users' characteristics, such as sending questionnaires to users (Resnick & Varian, 1997).

In general, recommender systems can be divided into four main categories (Adomavicius & Tuzhilin, 2005):

- **Knowledge-based Systems:** Depending on the needs and characteristics of the user, relevant suggestions are provided.
- **Content Based Recommender Systems:** These types of systems, through indexing and content analysis methods, keywords, tagging, graphing of relevant content and similar techniques, attempt to establish a conceptual relationship between the existing items and the item of user's interest.
- **Recommender Systems Based on Collaborative Filtering:** It is the most popular approach and it assumes that a popular pattern in the past is likely to be popular now. This approach benefits by making use of social media crowd sourcing and other recent socio-technological developments (Schafer, Frankowsk, Herlocker, & Sen, 2007).
- **Hybrid Based Recommender Systems:** Using both content-based and collaborative filtering techniques together.

This research has a special focus on collaborative filtering and matrix factorization model in recommender systems, and the final implemented method would be based on the matrix factorization model. Considering that the content-based recommender system is also a quite popular approach, this section will explain both methods together and various mathematical models will be introduced to evaluate the methodology of the proposed method and existing methods.

2.2 How a Recommender System Works

Before examining the various models used in the recommender systems, it is necessary to investigate the details of recommending and the recommender system to be precise to resolve any ambiguity. Based on the exact definition of the problem in this section and the defined symbols, the recommender system problem assumptions will be constructed.

In the definition of the user with the U symbol, the characteristics of a user, such as height, weight, age, sports interest, and so on, are marked with an A^U. The X^U symbol represents the behavior and specific user-specific information. X contains sensitive information, many of which are not readily available due to user inactivity in some areas. This information can include user-clicked links or individual comments about various items. A set of items (or existing elements on a site) is identified by the “I” symbol, and the properties of each component will be displayed with A^{item}. Determining the impact of user behavior on an item’s properties and the impact of item features on sales is very costly. Therefore, it is necessary to use methods that can help identify the implicit effect of an item on the user and also the implicit effect of users on the selection of items.

In a recommendation system, the method of examining information and proposing options (goods, services, movies, and books) is very essential; Additionally, collecting data from users has a significant impact on determining the cost and efficiency of the designed system. The information is extracted and stored in two implicit and explicit forms:

- **Implicit Information:** Based on the user's behavior on the system (site), information such as shopping, viewed links, videos, and comments are stored.
- **Explicit Information:** Many sites use survey and polling mechanisms in place of scoring to simplify the extraction of user information. This information can be used to rank existing items, score a movie or a good, and even list the selected items.

The Recommender system must provide a model based on the set of user attributes U, set of item characteristics I, and user feedback (privileges) that, by presenting new offers, can provide sales efficiency for an online store or other similar services. Although, the presentation of an intelligent model based on mathematical relationships to solve this issue may seem simple at first glance, but when we know that the information available from users is not always complete, as well as the many items that are not provided enough information about by users, the procedure becomes very complicated. One of the most critical problems is when the system cannot predict about the user's sensation about an item when he chose item 1 and does not chose item 2; Did he ever see this item and refuse to purchase it?

The Recommender systems, based on explicit and implicit information, are presented in the two categories of item recommending models (implicit information) and the user interest rate predictor models (explicit information). Of course, this categorization is not based solely on explicit and implicit information, but both categories of information can be used for both the recommendation and predictive system models (Resnick & Varian, 1997).

Table 2.1 shows the scores of people to movies based on the degree of interest. Anyone can grade a movie from 1 to 5. The goal is to predict a movie score that has not been viewed by a user named Steve so far. If we can guess Steve's point of the film, we will definitely be able to recognize that whether Steve likes the movie or not, and if the system offers a movie will he purchase it or no. Finally, a score of 4 or 5 would be a good benchmark to buy the offer for Steve.

Table 2.1: The recommender system in the predictive role of user interest rates (Horvath, 2012)

	Titanic	Pulp Fiction	Iron Man	Forrest Gump	The Mummy
Joe	1	4	5		3
Ann	5	1		5	2
Mary	4	1	2	5	
Steve	?	3	4		4

Table 2.2 shows user purchases, where each user's purchase is displayed with a value of one. The recommender system model for buying movies should predict that in case this movie is offered to Steve, will he buy it or not. This prediction must be done based on Steve's and other users' past purchases. In the following sections of this chapter, a brief review of the recommender systems will be addressed.

Table 2.2: The recommender system in the role of the proposer of the item (Horvath, 2012)

	Titanic	Pulp Fiction	Iron Man	Forrest Gump	The Mummy
Joe	1	1	1		1
Ann	1	1		1	1
Mary	1	1	1	1	
Steve	?	1	1	?	1

2.3 Recommender Systems Based on Knowledge

In such systems, user information and his needs play the most important role in determining the offer. A user's data is received and categorized in various domains and forms (Burke, Knowledge-based recommender systems, 2000):

- **Bounding Variables:** Determining specific bounds for goods and specific items. For example, buying a car worth less than $100,000 worth or introducing films produced between 2000 to 2010.
- **Determine Application:** Determine the abilities and characteristics of an item qualitatively. For example, a suitable car for a family or a drama movie.
- **Getting information through communication with the system:** In this case, it requires the processing of natural language and, of course, a verbal interface between the system and the user.

Depending on the user's profile and the properties of an item, two dependencies will be created. The first category is the affiliation of the items and their characteristics. For example, a family car cannot be less than a certain amount and weight. Another category is the dependency between user requests. For example, a safe car for all occupants and a price of more than $50,000 will not include all vehicles at a price of more than $50,000, and those that are not safe under the system's criteria will be removed.

Before examining existing and applicable items based on user-entered profiles, affiliations that affect one another or create new dependencies must be calculated and listed. For example, a family car must be large, and this large means that at least this car should have four passenger seats. A vehicle with four passenger seats must have 4 or more doors. Four passenger seats and four doors must have four airbags. And this list goes on like this.

After entering the requested item's specification and calculating the dependencies between item properties and the user's request, the problem should be solved by one of the ways to solve the problem of satisfying the constraints. One of the known methods in this area is the CSP method, or Constraint satisfaction problem (Constraint satisfaction method) (Koren, Bell, & Volinsky, 2009). Based on the constraints and the impact of constraints on one another, an option that does not violate any constraints (or the least constraints violation) should be selected. Another method is to use queries with logical relations "and" and "or" and provide results for the final choice to the user.

When the user is referring to the recommender system to select an item, the system should provide the implemented item concerning the existing items. Therefore, methods that work on the basis of existing samples and calculate the similarity of items are very acceptable. To calculate the similarity between existing items and user requests, the weight of each request based on the importance of the user must be determined. Of course, the price of a car may be much more important to a user than the color of a car, while for other user car security and color may have the same importance. Finally, the most relevant item will be selected according to the relation (2-1) to the requested items (Burke, Knowledge-based recommender systems, 2000).

$$Similarity(i, REQ) = \frac{\sum_{r \in REQ} w_r * sim(i, r)}{\sum_{r \in REQ} w_r} \qquad (2\text{-}1)$$

In relation (2-1), the similarity of item “i” is compared to the requested specification (REQ) and for each item a number between zero and one is specified. The most similar item on the user request has the largest number, compared to other items based on the similarity criterion (Burke, Evaluating the dynamic properties of recommendation algorithms, 2010). sim(i,r) determines the most similar item according to the user's request r and the weight of each request will be determined according to the user's value. For example, in car purchasing, security is definitely more important than car color, or at least for some users is more important.

Although for knowledge-based recommender systems, different methods are presented and introduced that their examination of all of them in this study is not possible due to the lack of similarity to the proposed solution, but it can be summarized in several sentences to one of the strongest learning machines called k-nearest neighbor (KNN). Based on the list of available items and request(s), the nearest neighbor in the n-dimensional space of the requested problem will examine the closest items to the user's requests. This method can better explore the search space and offer close proximity to user requests because it does not just decide on a request and considers similarities in all dimensions (Lathia, Hailes, Capra, & Amatriair, 2010).

2.4 Content-Based Recommender Systems

Based on the content of the text or texts written by a user and according to the user's past interests and current user interests, the recommender system should be able to provide suggestions in line with the user's interests. User-defined specifications can be explicitly stated (such as price, car or video production rate), or the program automatically extracts information from the user's written text (Lops, Gemmis, & Semeraro, 2011). One of the most common ways of extracting implicit information is to find the words used by the texts in the site's database and to formulate a feature vector based on keywords and check their existence (display with zero and one) based on the user's text. The TF-IDF model, based on the number of repetitions and the effects of repetitions in different texts, is very useful in extracting keyword attribute vector. The TF-IDF criterion of the TF multiplier, which is based on the effect of the number of repetitions of words, is obtained in the IDF, which is the word effect in all texts, as shown in (2-2) and (2-3). The criterion for each item on the site is a number indicating the user's interest in the item. The zero value indicates that the user is not interested in user-written texts. The higher the number is, the larger the user's interest rate (because of more repetition) of the desired item (Phelan, McCarthy, & Smyth, 2009).

$$TF(w,d) = \frac{freq(w,d)}{\max\{freq(w',d)|\, w' \neq w\}} \tag{2-2}$$

$$IDF(w,D) = \ log \frac{|D|}{|\{d \in D|w \in d\}|} \tag{2-3}$$

In the relationship (2-2), the TF value is calculated based on the number of repetitions of the word "w" (freq), in the text d and its ratio to the maximum number of repetitions of the word in that text. The value of the IDF is also calculated by dividing the total of the texts into the number of texts that contain the word, which can be calculated in relation (2-3). After finding the keywords and the number of them repeatedly, it is time to determine the items that the user has chosen in the past similar to them (regarding specifications). Accordingly, several different methods can be used, which in brief are some of them:

- **Cosine Vector Similarity:** From the internal multiplication, the vector of the item's selected attribute in the past and the current items and its division into the multiplication of these features is obtained. This defined relation is the same cosine angle of the two feature vectors, which, if its value is zero, it means the verticality of the two feature vectors and their total difference (Adomavicius & Tuzhilin, 2005). This criterion can be used to find similarity between the choices of a user and other users.
- **The k-nearest Neighbor Method:** In the item properties space, a search will be made based on the item's attribute that has been selected in the past and the

k-neighbors will be compared to select and suggest the same item to the selected item in the past. To calculate the similarity of simple criteria such as the Euclidean distance in the n-dimension, or the Mahalanobis distance, or even the cosine vector similarity, can be used (Horvath, 2012).

- **Rocchio's Method:** The Rocchio's model uses the positive and negative feedback provided by the user on each item. This model converges to a prototype that expresses the user's ideal item by repeating the algorithm presented on the feeds and the items. From now on, the suggestion to the user of the similarity of the items in the database to the ideal item will be found (Zanker, Felfernig, & Friedrich, 2011).
- **Machine learning:** One of the common methods is the use of learning machines to learn the relationship between the features of selected items and user feedback. Once the training process has been completed, the designed machine must be able to guess and predict the user's interest in the specified item by retrieving the new inputs of the same items. Learning machines, decision tree, Support Vector Machine are among the commonly used methods (Horvath, 2012).

2.5 Recommender Systems Based on Similarity Calculation

One of the most common methods for calculating the rating and similarity of various users' items is similarity methods. Based on the similarity of users or items, these methods find similar items and then offer the ones with most similarities as recommendations. The cosine similarity calculation method is one of the most famous of these models, which is fully explained in the next chapter and in the proposed method section. In this section, two methods of asymmetric user similarity and MMD will be explained.

2.5.1 Asymmetric User Similarity Model

Usually we are calculating the similarity to a triangular matrix of the side. If two u and v vectors are similar to 0.8, for the most similar relationships sim(u,v) = sim(v,u) = 0.8, it does not matter which first ones. For example, if a user "u" has a share of three items with a user "v," or two of the items most closely resemble each other, it does not matter whether checking "v" first or second. But in the asymmetric model, this story will generally be different, and the model will act to the same extent based on which target user. For example, if a user "u" has three items and "v" ten items and all three products u are in the v list, then the cosine computational method will be similar to those of 1.0, while this number is not the correct criterion for the similarity of the two models. Based on the look of the asymmetric model, "u" is completely similar to "v" since all "u" choices are made by the user "v," so probably the other seven are also of interest to "u." On the other hand, only 0.3 of the v choices with u is shared, and v may be very similar to other users and have many choices. As a result, it cannot be said that if the user "u" chooses a new product, it will also be attractive for v. This double-sided look will help you calculate the likeness of two users in a more realistic way. The phrase 2-4 shows how to calculate the similarity for both user's u and v. The number of first user items "u" has a lot of similarities. For above example, the value will be sim (u, v) = 1 and sim (v, u) = 0.3. The number of items per user will have a direct impact on the determination of similarity, and if the number of subscriptions of the two users is high over the whole of the first user items, similarity will also be high (Pirasteh, Hwang, & Jung, 2015).

$$sim(u, v) = \frac{|I_u \cap I_v|}{|I_u|} \tag{2-4}$$

The method presented in Expression 2-4 is not an appropriate analogy, since the consideration of the number of subscriptions alone may not reflect the exact relationship and similarity. The Expression 2-5, by adding the amount of subscription to the total items, attempts to reduce the impact of the number of items of a particular user and increase the effect of the proportion of similarity of each user to the total of the two user items (Pirasteh, Hwang, & Jung, 2015).

$$sim(u,v) = \frac{|I_u \cap I_v|}{|I_u|} * \frac{2 * |I_u \cap I_v|}{|I_u| + |I_v|} \tag{2-5}$$

The MSD method is a symmetric method for calculating similarity, which, unlike two expressions 2-4 and 2-5, is determined by the scores of users. The MSD calculation method is visible in the expression 2-6 (Shardanand & Maes, 1995). Finally, using the MSD symmetric method and the asymmetric method introduced in Expression 2-5, a final composition is presented in Expression 2-7, which has the characteristics of the effect of the number of items in it, and, of course, the rating of users to the goods is also effective. The value of L in Expression 2-7 is a threshold defined to normalize MSD output values, which can be modified based on experience and error testing. The authors of this paper considered the default value for L 16 (Pirasteh, Hwang, & Jung, 2015).

$$MSD(u,v) = \frac{\sum_{p \in |I_u \cap I_v|} (r_{u,p} - r_{(v,p)})^2}{|I_u \cap I_v|} \tag{2-6}$$

$$Asim(u,v) = \frac{L - MSD(u,v)}{L} \frac{|I_u \cap I_v|}{|I_u|} * \frac{2 * |I_u \cap I_v|}{|I_u| + |I_v|} \tag{2-7}$$

2.5.2 Mean Measure of Divergence Similarity

The MMD method, like similar methods, has a special emphasis on subscribing to two-user products, with the difference that the criterion of subscribing to scoring is valid.

For example, if two users with similar ratings of 3 are for two products, they will be more similar until the two users have voted for a product. In other words, the focus in this way has changed from goods to points, and the number of similar points in the users has the same behavior and quality. The phrase 2-8 shows how to calculate the MMD. The variable “r” in the following statement represents the score from 1 to 5 (the lower and the upper score limit for a rating site). The theta value also includes the number of ranks the user enters with the value of r (Mahara, 2016).

$$sim(u,v) = \frac{1}{1 + (\frac{1}{r} * \sum_{i=1:r}\{(\theta_u - \theta_v)^2 - \frac{1}{|I_u|} - \frac{1}{|I_v|}\})} \tag{2-8}$$

The MMD method is successful in calculating the same behavior of users, but, as it has been said, it is not able to consider similar products. For this reason, the authors introduced the proposed method to combine and use the two jaccard models and cosine similarity with the implemented method. In the jaccard method, the number of item subscriptions expresses similarity, and thus two users who have more items with each other will be more important. In the case of cosine similarity, we also know that the similarity of users' points of view on direct and indirect affluent goods has a negative effect. The expressions 2-9 and 2-10 show the jaccard calculation formula and the final version of cjacMD, which combines cosine, jaccard, and MMD (Shardanand & Maes, 1995).

$$sim(u,v)_{Jaccard} = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} \tag{2-9}$$

$$\begin{aligned} & sim(u,v)_{CjacMD} \\ &= sim(u,v)_{Jaccard} + sim(u,v)_{Cosine} \\ &+ sim(u,v)_{MMD} \end{aligned} \tag{2-10}$$

2.6 Recommender Systems Based on Collaborative Filtering

In collaborative Filtering methods, there are two dominant categories, the former being defined and presented based on neighborhood models, and the second one is includes Latent Factor methods (Resnick & Varian, 1997). In the neighborhood-based methods, the goal is to calculate the relationship between users or the relationship between items. In this method, the rating of an item or a user is determined based on their neighbors. In contrast to neighboring methods, methods that are based on the Latent Factor tend to compute and find hidden relationships between users and items (Bellogin, Cantador, Diez, Castells, & Chavarriaga, 2013). These Latent Factors do not have a precise definition or constant behavior, but are extracted and prepared using methods such as matrix decomposition as new dimensions and a new interface matrix between users and items. In the next section, firstly the methods that act based on the neighborhood of the item will be examined, and then the Latent Factor methods will be introduced and presented.

2.6.1 Matrix Factorization

The main purpose of the matrix factorization method is to generate two matrices of the item and the user in such a way that the multiplication of these two matrices is the rank matrix (which represents the rank of each user per item). This technique has been well known in recent years due to the combination of acceptable scalability and accurate prediction accuracy. The most suitable data for matrix factorization are data with explicit and high-rated feedback that includes explicit inputs in which users have expressed their interest in the items. For example, Neflix collects star ratings for movies, and Tivo users specify their preferences and interests to television shows based on the Thumbs-Up and Thumbs-Down buttons (similar to likes or dislikes).

Typically, explicit feedback results in a sparse matrix, since each user is likely to only score a small percentage of items (Parambath, 2013).

One of the matrix factorization capabilities is that it enables a combination of additional privileges. When explicit feedback is not available, the recommender systems can deduce the preferences and interests of users based on implicit feedback, which indirectly reflects the views and reflects the behavior of users, including the purchase date, site visit history, search patterns, and even mouse movements. Implicit feedback typically indicates the presence or absence of an event.

Matrix factorization models map users and items into a shared Latent Factors lying in the f dimension, such that the user-item interaction is modeled as the internal multiplication in that space. Accordingly, each item i is associated with a $q_i \in R^f$ vector and each user u corresponds to a vector $p_u \in R^f$. For an item given i, q_i elements are expanded and modified to determine the effect of the underlying factors on its positive or negative path. For a given user u, the p_u elements show the amount of user's desire to the items with positive and negative values. The multiplication of the dot $q_i^T p_u$represents the interaction between the user "u" and the "I" item. This approximation of the user u to the item i shown with r_{ui} is estimated as follows (Parambath, 2013):

$$\hat{r}_{ui} = q_i^T p_u \qquad (2\text{-}11)$$

The main challenge is to calculate the mapping of each item and user to vectors$q_i, p_u \in \mathbb{R}^f$. After calculating the proposing system of this mapping, it can simply estimate the score that the user gives to each item based on relation (2-11). Such a model is close to dividing the Singular Value Decomposition unique value, which is a good way to identify the semantic factors involved in data retrieval. The use of SVD in the collaborative refinement domain requires the user-item rating score. This problem is usually encountered in cases where a large portion of the values is lost because of the lack of user-item matrix sparse. The conventional SVD is unclear when knowledge of the matrix is not complete. Also, the uncertainty of several relatively low-level inputs strongly affects the system's risk of over-fitting (Parambath, 2013) (Adomavicius & Tuzhilin, 2005).

Recent systems are based on relying on the assignment and filling of lost scores and clumping the scoring matrix. Although this assignment can be very costly, it will increase the amount of data. In addition, inaccurate assignment can significantly distort information. Also, most of the recent work offers a direct modeling of the observed scoring, and thus prevents over-fitting due to regularization. In order to learn the vectors of the p_u, q_ifactor, the square error system (relation (2-12)) minimizes the known scoring set (Parambath, 2013).

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2) \tag{2-12}$$

The proposed system learns the model based on matching previously reviewed scores. Although the goal is to generalize previous scores in line with future predictions and unknown scores; therefore, this system should prevent the fitting of the data observed by regulating the parameters learned that their amounts are fined. The parameter λ controls the level of regularity and is usually determined by Cross-validation. Ruslan Salakhutdinov and Andriy Mnih (2007) presented a probabilistic function for regularization.

The two approaches to minimizing the relation (2-12) are the randomized descent gradient as well as the least squares of variables that are discussed below.

2.6.2 Random Descent Gradient

Simon Funk introduced an optimal randomized descent gradient optimization algorithm that runs on all training rankings. For each instruction given, the system predicts the r_{ui} system and calculates the corresponding error (Parambath, 2013) (Gemulla, Nijkamp, Haas, & Sismanis, 2011).

$$e_{ui} \overset{\text{def}}{=} r_{ui} - q_i^T\, p_u \tag{2-13}$$

Then the parameters are updated with the value of γ (which is a normalization parameter) and in the opposite direction of the gradient.

$$q_i \longleftarrow q_i + \gamma\,.\,(\,e_{ui}.p_u - \lambda\,.\,q_i)$$

$$p_i \longleftarrow p_u + \gamma\,.\,(\,e_{ui}.q_i - \lambda\,.\,p_u)$$

2.6.3 Alternation Least Square (ALS)

Since $p_u, q_i$are unknown, relation (2-13) is not a convex relation. However, if we put one of them in a constant way, the optimization problem becomes a second-degree relation and can be solved optimally. Therefore, methods based on the least squares of the variables, rotate between fixing p_u and qi.

When p_u is proved, the system reconstructs qi by solving a least squared problem, and vice versa. This ensures that relation (2-13) is decreasing in each step until convergence (Parambath, 2013).

Although in general the randomized descending gradient of the least squares of the variable is simpler and faster, the ALS is desirable in at least two cases. The first case is where the system can use parallelization. In the ALS, the system calculates each q_i as non-dependent, and to calculate the effect of other item factors, and also calculates each p_u independently of other user factors. This potentially broadens the parallelization of the algorithm. The second concern is regarding the centralized systems on implicit data. Because the training set cannot be sparse, the implementation of the loop on any training sample alone is not feasible. ALS can handle these issues efficiently.

2.6.4 Add Bias

One of the advantages of the matrix decomposition approach in collaborative filtering is its flexibility to deal with different data and other program requirements. This requires attention to the relation (2-12) in the learning process. Relation (2-12) attempts to capture the interaction between the users and the items that produce different scores. Although most of the variation observed in the scores are based on the effects of the user and item, and independent of any other interactions. For example, shared data refinement shows great systematic tendencies for some users to give a higher rating than the rest, and also show greater tendency for some items to receive a higher score than the rest. As such, some of the products are widely better than other products.

Therefore, it is not wise to explain the value of a complete score based on an interaction as $q_i^T p_u$. Instead, the system tries to detect a portion of these values that the user scores individually or identifies the values that the bias can explain for the items (Relation (2-14)) (only the real interaction of the part Latent Factor modeling). A first-order estimate of the bias involved in r_{ui}'s score is given by the following equation (Parambath, 2013):

$$b_{ui} = \mu + b_i + b_u \qquad (2\text{-}14)$$

Here, the observed score is divided into 4 elements. 1) Global average, 2) Item's bias, 3) User's bias, 4) User interaction with an item. This causes each element to explain only the part of a corresponding signal. Systems based on minimizing the error function will be trained (equation (2-15)) (Parambath, 2013).

$$\min_{q^*, p^*, b^*} \sum_{(u,i)\in K} \begin{matrix} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \\ \lambda \left(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2\right) \end{matrix} \qquad (2\text{-}15)$$

2.6.5 Additional Input Resources

Typically, when many users offer a low rating and ranking, it is difficult to reach a general conclusion based on their tastes and interests. One way to overcome this problem is to add additional resources to information about users.

Recommender systems can use tacit feedback to evaluate and evaluate users' preferences and preferences. In fact, they can provide an explicit rating, without considering and collecting the user's desire. A retailer can use their customers 'purchases or from customers' visit dates to learn user desires. In addition, they can use them to estimate the ratings that may be offered by buyers.

For simplicity, consider an item with the implicit feedback of zero and one. N(u) represents a set of items that the user prefers to buy them or rate them in a way. In this way, the system is made up of users based on the items they implicitly prefer. Here is a new set of item factors. The item i is expressed by $x_i \in \mathbb{R}^f$. Similarly, a user who points to items within N(u) is identified by the following vector (Parambath, 2013):

$$\sum_{i \in N(u)} x_i$$

Normalizing this collection is usually helpful. For example, it can be normalized to the following equation (Parambath, 2013):

$$|N(u)|^{-0.5} \sum_{i \in N(u)} x_i$$

Other sources of information identify user attributes, for example, demographic or demographic data. Again, in order to simplify, consider the positive and negative attributes that the user has with the set of features A(u). This feature set can include gender, age, national number, economic level, and other characteristics. Based on a unique factor vector $y_a \in \mathbb{R}^f$ corresponding to each attribute, a user is identified by a set of its properties (Parambath, 2013):

$$\sum_{a \in A(u)} y_a$$

The matrix decomposition model should integrate all the signal sources that are enhanced by the presence of the user (Relation (2-16)) (Parambath, 2013).

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a] \qquad (2\text{-}16)$$

2.6.6 Temporal Dynamics

So far, the proposed models have been introduced statically. In fact, the image of the product and its popularity are continually changing and emerging as a new product. Similarly, users' desire can evolve and can be changed. Therefore, the system should consider the effects when it shows dynamism.

The matrix decomposition approach is appropriate for applying time effects and can increase accuracy with respect to it. Matrix decompositions by considering distinct phrases allow the system to have distinctly different behaviors at different times. Specifically, the following phrases change over time:

- Items orientations $b_i(t)$
- User orientations $b_u(t)$
- Also, the users preferences $p_u(t)$

The first-time effect is that the popularity of an item can change over time. For example, movies can be added to or removed from popular movie listings based on the impact of external factors. Therefore, in this b_i model, which represents the orientation of an item, it is considered as a function of time. The second time impact allows users to change their scores over time. For example, a user rating a particular 4-star movie can later change its score to 3 stars. It should also be noted that the evaluator's identity could change over time. In this model, the parameter b_u is also designed as a function of time (Parambath, 2013).

The temporal dynamics proceed as stated. They continue to affect the user's desires and the interaction between users and items. Users change their interest over time. For example, a fan of psychological-style plays may become a fan of criminal movies next year. Similarly, people's perspective about actors and directors can change over time. These issues have been applied to this model, considering user factors as a function of time. But unlike people, items are static and they do not change over time. Therefore, the relation (2-14) can be rewritten with the application of temporal dynamics (2-17) (Parambath, 2013).

$$\hat{r}_{ui} = \mu + b_i(t) + b_u(t) + q_i^T p_u(t) \qquad (2\text{-}17)$$

2.6.7 Input with Different Reliability Levels

In some settings, all observed scores do not have the same level of confidence and weight. For example, widespread advertising for a particular item cannot properly reflect the features of that item. Similarly, the system may face users who seek to rank differently for that product in order to advertise specific products.

Another example relates to a system made with tacit feedback. In such systems that are currently interpreting the behavior of the user, it is difficult to determine precisely the priorities and desires of the user. Hence, the system works with a binary representation that offers both “willingness to product” and “unwillingness to product” state. In such cases, determining the confidence coefficient of these estimates is a valuable issue. The determination of the confidence coefficient can be based on existing numerical values that indicate the frequency of the actions. For example, how long the user spent viewing a show or with what frequency bought an item. These numerical values represent the degree of assurance in each observation. There are various factors that do not have a particular impact on the user's perspective; they may cause a momentous event. Though events that occur alternately have reflect on user feedback with higher probability.

The matrix decomposition model can accept various levels of confidence so that it allocates less weight to less obvious observations. If the confidence of r_{ui} is shown with c_{ui}, then the function model will result in the following cost (Parambath, 2013):

$$\min_{q^*,p^*,b^*} \sum_{(u,i)\in K} \begin{matrix} c_{ui}(r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \\ \lambda\,(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \end{matrix} \tag{2-18}$$

2.7 Use of Genetic Algorithm for Matrix Factorization

A genetic algorithm is a tool by which the machine can simulate the natural selection mechanism. This is done by searching the problem space to find a superior answer, and not necessarily optimal. The genetic algorithm can be called a general search method that mimics the laws of natural biological evolution. In fact, genetic algorithms use Darwin's natural selection principles to find the optimal formula for predicting or matching patterns. Genetic algorithms are often a good option for regression-based prediction techniques. The genetic algorithm, which is the method of optimization inspired by the nature of the living organism that can be categorized as straightforward and random search as a numerical method. This algorithm is a repetition-based algorithm, and its initial principles have been adapted from genetic science as previously mentioned and invented by imitation of some observed processes in natural evolution. This algorithm is used in a variety of problems such as optimization, identification, and control of the system, image processing, and hybrid problems, the determination of topology and the training of artificial neural networks and decision-making systems (Salomon, 1996).

The Genetic algorithm as an optimization computational algorithm, with consideration of a set of spatial points in each computational recurrence, effectively searches for different areas of the answer space. In the search mechanism, though, the value of the objective function of the entire solution space is not computed, but the calculated value of the objective function for each point is in the mean value of the target function for each point and in the averaging of the target function in all sub-spaces where that point is dependent It is interfered with, and these sub-spaces are statistically equated in terms of the objective function. This mechanism is Implicit Parallelism. This process leads to the search for space in the regions where the mean of the statistical function of the objective function is high and the possibility of an absolute optimal point in them is greater. Because in this method, unlike the replication methods, the search space is searched comprehensively, there is less possible convergence to a local optimal point (Srinivas & Patnaik, 1994).

In the genetic algorithm, a set of design variables is encoded by strings of Fixed Length or Variable, which in their biological systems refers to them as chromosomes or individuals. Each strand or chromosome shows a response point in the search area. The structure of the strings, a set of parameters that is represented by a particular chromosome, is a genotype and its decryption is a phenotype. Each repeat step is called generations and sets of responses in each generation called the population. Genetic Algorithm By providing a user-user matrix and an item-item, the R-matrix is split into two U and I matrices, so that the multiplication of these two matrices has the least error in rebuilding the training set. Genetic Algorithm Initially generates random values for both matrices and improves parameters. By reducing the RMSE error, the training set goes up to the global optimal problem. Finally, the best answer is to contain two user-user and item-item arrays, each multiplied by a new R 'matrix, which includes the unknown values of the R matrix, and has new suggestions within itself.

2.8 Cosine Similarity Criterion for the Item-based Similarity

The collaborative filtering method, as previously discussed, is based on other users' feedbacks on finding the user-favorite item. In identifying the user x’s favorite items, two major categories of solutions, one based on the user's behavior and the other one based on the item's profile, are proposed and presented. The method which makes suggestions based on the user's behavior finds the most similar user (s) based on the user’s profile characteristics and introduces their interests as a suggestion to the user x. The basis of the item profile-based solution is to focus on the items which the user chooses and is similar to other items regarding technical specifications and appearance. The relations (2-19) and (2-20) represent the method of calculating the most common user based on user x and the most similar item based on the item selected by user x (Schafer, Frankowsk, Herlocker, & Sen, 2007). The *sim* function in relations (2-19) and (2-20), means finding the similarity between two users or two items. In (2-4) and (2-5), the value of k, which is the sigma subfield, represents the size of U' or I' where U' represents users which are similar to the user U_i and I' represents goods similar to I_u .

$$N_i^{u,k} = argmax_{U'} \sum_{\substack{v \in U', v \neq u \\ U' \subseteq U_i, |U'| = k}} sim(u, v) \tag{2-19}$$

$$N_u^{i,k} = argmax_{I'} \sum_{\substack{j \in I', j \neq i \\ I' \subseteq I_u, |I'| = k}} sim(i, j) \tag{2-20}$$

Based on the relations (2-19) and (2-20) for each user and each item, we will have choices that will either suggest an $sim_{cv}(i,j)$ item or introduce users similar to x. Relying on a user based on the similarity criterion is not a reliable solution to predict the future behavior of the user x; with the help of the k-nearest neighbor method, one can search for and find the average behavior of the items or similar users in the multidimensional space of the problem and offer the outcome as a suggestion. The relationship (2-21) and (2-22), using averaging on the k-neighbor of the user or the proposed item, will give a more accurate result to the output (Schafer, Frankowsk, Herlocker, & Sen, 2007).

$$\phi_{ui} = \frac{\sum_{v \in N_i^{u,k}} sim(u, v)}{k} \tag{2-21}$$

$$\phi_{ui} = \frac{\sum_{j \in N_u^{i,k}} sim(i,j)}{k} \tag{2-22}$$

For a closer look at what has been told about the nearest neighbor and the similarity criterion in collaborative filtering based on the neighbor, the example of the films watched by four users is plotted and evaluated according to the ratings each user has given to the films. Table 2.3 shows the similarity of each film based on its predetermined characteristics and based on the trigonometric vector criteria. On the other hand, Table 2.4 shows the similarity of users' selections with one another, namely, the calculation of the relationship (2-21).

Table 2.3: Similarity of films based on the calculation of the relationship (2-22) (Sarwar, Karypis, Konstan, & Riedl, Item-based collaborative filtering recommendation algorithms, 2001)

$Sim_{cv}(I,j)$	Titanic	Pulp Fiction	Iron Man	Forrest Gump	The Mummy
Titanic	1.0	0.87	0.67	0.82	0.67
Pulp Fiction	-	1.0	0.87	0.71	0.87
Iron Man	-	-	1.0	0.41	0.67
Forrest Gump	-	-	-	1.0	0.41
The Mummy	-	-	-	-	1.0

Table 2.4: Similarity of users based on the calculation of the relationship (2-21) (Sarwar, Karypis, Konstan, & Riedl, Item-based collaborative filtering recommendation algorithms, 2001)

$Sim_{cv}(u,v)$	Joe	Ann	Mary	Steve
Joe	1.0	0.75	0.75	0.87
Ann	-	1.0	0.75	0.58
Mary	-	-	1.0	0.58
Steve	-	-	-	1.0

By calculating the relation (2-21) for a user named Steve and items Titanic and Forrest Gump, we see that the user may be more interested in the Titanic movie.

$$N_{Titanic}^{Steve,2} = \{Joe, Ann\}, \varphi_{ST} = \frac{a_{cv}(S,J) + a_{cv}(S,A)}{2} = \frac{0.87 + 0.58}{2} = 0.725$$

$$N_{ForrestGump}^{Steve,2} = \{Ann, Mary\}, \varphi_{ST} = \frac{a_{cv}(S,J) + a_{cv}(S,M)}{2} = \frac{0.58 + 0.58}{2} = 0.58$$

The method used was the similarity calculation method based on the similarity of the users (nearest neighbor of the relation (2-21)). Another way, as suggested before, is the similarity method based on the similarity of item properties. In the example below, you will see that this method also gives a vote for the Titanic film to introduce to Steve.

$$N_{Steve}^{Titanic,2} = \{PulpFiction, IronMan\}, \varphi_{ST} = \frac{a_{cv}(T,P) + a_{cv}(T,I)}{2} = \frac{0.87 + 0.67}{2}$$
$$= 0.77$$

$$N_{Steve}^{ForrestGump,2} = \{PulpFiction, IronMan\}, \varphi_{ST} = \frac{a_{cv}(F,P) + a_{cv}(F,I)}{2}$$
$$= \frac{0.71 + 0.41}{2} = 0.56$$

Finding one or more items and to offer them to a user is not the only aim consistently. Sometimes knowing the user's potential score to a movie, product or one of the provided services will be very effective in improving or modifying the existing system. In collaborative filtering, methods are presented based on users' average behavior in choosing similar items and also the average points of the items by users, to guess the score of an item by a particular user. Relationships (2-23) and (2-24) calculate the user u's score to the item I based on the similarity of the two users (according to the selected items) (Sarwar, Karypis, Konstan, & Riedl, Item-based collaborative filtering recommendation algorithms, 2001).

$$\varphi_{ui} = \overline{\varphi_u} + \frac{\sum_{v \in N_i^{u,k}} sim(u,v) \,.\, (\varphi_{vi} - \overline{\varphi_v})}{\sum_{v \in N_i^{u,k}} |sim(u,v)|} \tag{2-23}$$

$$\overline{\varphi_u} = \frac{\sum_{i \in I_u} \varphi(u,i)}{|I_u|} \tag{2-24}$$

The relationship (2-24) is how to calculate the average user rating for all of his selected items. Relationships (2-25) and (2-26) also calculate an item's score for a particular person based on similar items (Sarwar, Karypis, Konstan, & Riedl, Item-based collaborative filtering recommendation algorithms, 2001).

$$\varphi_{ui} = \overline{\varphi_\imath} + \frac{\sum_{j \in N_u^{i,k}} sim(i,j) \,.\, (\varphi_{uj} - \overline{\varphi_\jmath})}{\sum_{v \in N_u^{i,k}} |sim(i,j)|} \tag{2-25}$$

$$\overline{\varphi_\imath} = \frac{\sum_{u \in U_i} \varphi(u,i)}{|u_i|} \tag{2-26}$$

The trigonometric vector cannot properly measure the variance and average scores in calculating the similarity between two users or two items. Therefore, the cosine-correlation similarity criterion shown in (2-27) and (2-28) is respectively used for the user and item (Sarwar, Karypis, Konstan, & Riedl, Application of Dimensionality Reduction in Recommender System - A Case Study, 2000).

$$sim_{pc}(u,v) = \frac{\sum_{i \in I_{uv}} (\varphi_{ui} - \overline{\varphi_u})\,(\varphi_{vj} - \overline{\varphi_v})}{\sqrt{\sum_{i \in I_{uv}} (\varphi_{ui} - \overline{\varphi_u})^2} * \sqrt{\sum_{i \in I_{uv}} (\varphi_{vi} - \overline{\varphi_v})^2}} \tag{2-27}$$

$$sim_{pc}(i,j) = \frac{\sum_{u \in U_{ij}} (\varphi_{ui} - \overline{\varphi_u})\,(\varphi_{uj} - \overline{\varphi_\jmath})}{\sqrt{\sum_{u \in U_{ij}} (\varphi_{ui} - \overline{\varphi_\imath})^2} * \sqrt{\sum_{u \in U_{ij}} (\varphi_{uj} - \overline{\varphi_\jmath})^2}} \tag{2-28}$$

The similarity function in relations (2-27) and (2-28), based on the Pearson correlation similarity criterion for users and items, can be used. The behavior of the similarity criterion in the above relationships is based on the effect of the average behavior of users (items) similar to the user u (item i) and using relationships (2-27) to (2-28).

The implemented method can predict the interest of users in various items, and these predictions are based solely on the history of the user's behavior and regardless of the intermediate relationship of the factors determining the ratings, or causes, such as how to attract users.

2.9 Holonic Multi-agent Systems

A multi-agent system will be able to complete a project by defining different tasks and events for different agents with a mutual yet public purpose. For example, in a smart home, each of the home appliances can be an agent or some controlling agents may be in charge of controlling several parameters at homes such as the temperature or electricity usage. Although the tasks have a direct relation to the number of intelligent agents and controlling factors, the entire project is one, and the goal of the whole system is to reduce energy consumption in the smart home. Hence, with the clever design of a multi-agent system, it will be possible to decentralize the task and provide a quicker and more accurate functioning (Hoek & Wooldridge, 2008).

Multi-agent systems are used to manage and complete complex projects. In such projects, the controlling and influencing factors are more than one, so each agent will be responsible for reviewing, controlling, and performing the assigned tasks. Another important feature of multiprocessing is its interactive behavior. Usually, a problem in the world of technology will be taken into the domain of multi-agent systems when the segmented sectors will also need to interact with each other. For instance, remember the example of a smart home. If all of the controllers can be managed with a central controlling system, maybe less time and cost will be needed compared to a multi-agent system. However, the problem will be taken to the domain of multi-agent systems when exact implementation of tasks of each section is required individually and also the interaction between all the sections is a must to complete the project. For example, if the central control system is constantly checking the conditions while the whole smart home is to being cabled, the real-time functioning of the system will be very costly. But if each autonomous agent independently examines the possible condition changes, it may result in a lower cost for the whole system. The main challenge remains unresolved, and that is the discussion of the interaction of intelligent agents. What is the reason for the need for interaction? The answer, however, is very simple; when an air conditioner is supposed to reduce force a few degrees due to the high power consumption of the refrigerator in the other room, it will be necessary to have a simple exchange of messages and making decisions.

The principles of independence and exchanges are two important principles in the formation of a multi-agent system. Figure 2.2 shows a representation of a hypothetical model of multi-agent systems. In the system below, there are four independent agents that communicate with each other, each of which identifies the output or input of the other agent. In the following mode, there is no factor to control other factors (Fischer et. al., 2003).

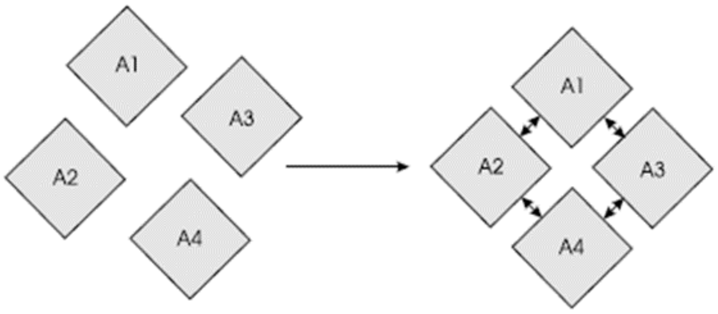


Figure 2.2: A multi-agent system with four agents (Fischer, Schillo, & Siekmann, 2003)

The word Holon was first used by A. Koestler in 1967, which means a structure composed of similar units. In the process, the Holonic became an entity that was made up of parts called the atomic. Each atom contained a series of tasks and processes that worked independently, but to eventually complete the project, there was a need for collective engagement. The difference between a Holonic and a multi-agent system is the existence of an observer in the system. This observer has the task of controlling exchange messages, checking errors, setting up the sub-agents, and stopping them. This Holonic observer is designed to increase the optimal interaction at the level of the Holonic so that each agent will call/stop at the required time, and each event is urgently called to the position when needed. Figure 2.3 shows a depiction of a Holonic demonstration model in which four agents are controlled by the central A supervisor (Fischer, Schillo, & Siekmann, 2003) (Gaud, Gechter, Galland, & Koukam, 2007)

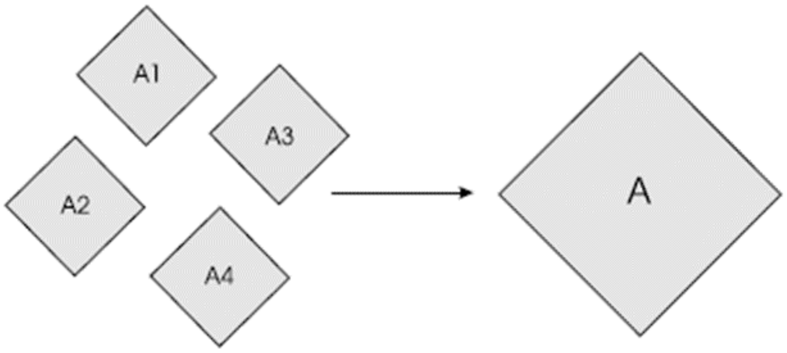


Figure 2.1: Representation of a hypothetical Holonic with 5 agents (Fischer, Schillo, & Siekmann, 2003)

Recommendation systems are usually single-agent systems that, based on user-rated scores for items, try to make predictions of other products of interest to the users. If it is possible to implement a system in a single-agent form, the implementation of a multi-agent version of the same system will not only burden the addition of extra complexity to the system, but it will also require more computational costs and more memory space (Fischer, Schillo, & Siekmann, 2003) (Rodriguez, Gaud, & Galland, 2014).

At first glance, there may be no need to use a Holonic or multi-agent system to design a recommendation system. However, after leaving the frame of academic experiments and entering the implementation of algorithms on real-world servers and electronic commerce websites, it will be a completely different issue. At any given moment, tens of thousands of new rating points are registered to different items by users on a website like Amazon. On the other hand, several new products (items) will be added by the vendors every minute, and most importantly, all data is not kept on a single server. In other words, we do not have to deal with a system that contains training and test information, and we are dealing with a large amount of previous information, current processing, new goods, and, of course, new purchases. The more different and independent tasks of a recommendation system extracted, the higher the chance of designing a high-quality Holonic with more processing speed, higher accuracy, and performance.

2.10 Review Repetition

In this chapter, the recommender systems and the issue of advancement of the system were introduced and discussed. Various methods have also been mentioned for the design and construction of recommender systems. One of the successful models is the matrix factorization method and similarity and neighborhood algorithms, which make the initial prediction and improvement of the decomposed matrix based on the methods mentioned above very important. Finally, Holonic multi-agent systems has been explained briefly. In Chapter 3, the proposed method for solving the recommender system problem will be introduced and examined, using a Holonic cosine similarity algorithm in a distributed manner.

Chapter 3

PROPOSED METHOD

3.1 Introduction to the Implemented Method

With the expansion of the Web and social networks, the world is encountering with electronic commerce and online shopping service providers with millions of records from tens of thousands of users who have commented on hundreds of thousands of merchandises. In such a large volume of data, the use of traditional algorithms with high processing time is practically impossible. For instance, Amazon stores more than 10,000 sales and more than a thousand comments every second in each state of the United States. It is not only time-consuming but also cost-effective regarding computational load to process all the existing data -which are generally in the size of several Gigabytes- and then expect to use the results for a new user and suggest a new recommendation.

To design an efficient, high-speed and real-time system, new methods that can update their information without having to re-learn all of data are needed. These methods should only decrease the quality of predictions and suggestions at the minimum level if they do not increase. It is only by using a quick and real-time system that it is possible to create a new list of offers by entering a new product or user, causing the sale of new goods, create interest for the new user, and, of course, extending the user's time spent on the site. In this chapter, we will first discuss the agents and events in the suggested multi-event system, then the parallel computing side of the work will be explained.

To design a Holonic system, it is necessary to design and identify the agents and events of the system. An event in a Holonic system involves an out-of-system event (system input) or an event that has been deliberately compromised by a system. For example, in the case of Smart House, the system will be logged in to the courier consumption event by arriving at the period of the energy consumption peak. Or, in another example, when the monthly consumption reaches the critical point, the system calls the maximum saving event. Obviously, in each defined event, certain operations and processes occur that is specific to that event. The definition of the agent is also clear for the Holonic system, which includes its main tasks in the various events. For example, for a cooler, the cooler temperature is reduced / increased, for the controller of the lights, the lights are off / up, and so on.

3.2 Agents and Events in a Multi-event System

In order to be more precise on defining the structure of the system, the implemented model is named as a distributed multi-event system rather than a multi-agent system. The reason behind this clarification is the importance of the two principles of multi-agent systems: Independence and Interactions of the agents. In the implemented model, the canon of the work is focused on the roles of the events while the agents are playing the role of the functions in the background. In another word, the model is benefiting from the multi-event structure of the system to provide the solution. Also, the model can function properly without any need of the agents to interact with each other.

3.2.1 “NewEntry” Event

As previously mentioned, in a recommendation system with a huge amount of rating records that is updated at any time, it is not possible to study all records at intervals from the beginning to the moment. It is necessary to find methods that require minimal processing and calculations during the process for the system to be updated. That is why the main event of the implemented method is a new Entry event. This event will be activated whenever a new record has entered the system that includes a user rating for a product i. The purpose of creating this event is to calculate the favorable item similar to the item i. In the next step, users who have already selected the same item similar to the item i, will be selected by the cosine similarity approach and the new item will be introduced to them. By launching the program and entering the first rating record, similar items are found and offered to users. As the program receives more records, the calculated similarities will be more accurate. Therefore it is expected that as the time goes on, the proposed system error will be reduced due to an increase in the dataset. In the defined event, the intent and purpose is to find items, similar to the item I that is rated by user u, and then introduce it to him. Hence, if for example the user u, record a rating for the item on the first day, then the items similar to itself will be introduced to him. But if the user does not give a new rating in the next days, then the list of the recommended items will not be updated. The lack of updating the list is because this event should generally run once by the related agents.

3.2.2 "OnUpdate" Event

A new event named onUpdate has been introduced to avoid the problem of not updating without a new rating. In this event, it is tried to find n similar users (n is a small number between 1-10), after each new rating has been given by the user u, and they will also be updated. Each time the event is called, the corresponding agent first has the task of finding n similar users and then, based on the new list, find similar items and updates the list if needed. To make the system update calculations do not slow down the system, after reviewing every m records and every m times of calling newEntry, the onUpdate agent will be called (m is a number between 1-10). Therefore, every couple of seconds, the old users will also be confronting a change and updating the list, if they find new users.

3.2.3 “ItemSimilarity” Agent

There are two categories of agents for calculating the list of recommended items, updating and calculating prediction errors controlled by the central supervisor, which is itself an agent. The first agent, ItemSimilarity, is responsible for calculating the similarity of items and finding similar products for each user; thus, it identifies the recommended items for each user. This agent also specifies the similar users for updating inside the onUpdate event. Each agent can operate in any event or remain inactive. The ItemSimilarity agent operates on both events. In the NewEntry event, this agent first uses relation 2-28 to identify the items of the same type as the item I in the new record. The number of similar items can be between 0 and I (the total number of items). The similarity of the items, as stated above, is determined by the user's interest and the item's features are neglected. This will also increase the processing speed. After specifying the number of similar items, the current user u who was interested in item I, will be introduced to a list of similar items. Note that the output of equation 2-28 or the cosine similarity is a number between 1 and 5, depending on the threshold (value 3), its high values will be suitable as a recommendation. The algorithm 3-1 is a demonstration of ItemSimilarity agent.

In the OnUpdate event, the agent ItemSimilarity will also use the relation 2-28 to calculate similar users to the user i. Then all of the n users that have the highest value, meaning that they have the most similarity to the user u, are selected. In this event, after assigning the similar users, the value is taken to the output to be passed by the supervising agent to the computing agent. Each agent can be called once or multiple times and can be killed afterward. By doing so, each agent will even be able to compute and find the similar products with the least computational system and with sufficient level of access to the server. Algorithm 3-1 is a demonstration of the behavior of the agent ItemSimilarity.

```
Agent Item_Similarity

Event : new_Entry
Inputs : Scores, Users,Items
Outputs : Predicted_Scores
For each new score in Scores
  I = current item
  U = current user
  Similar_Items 🡠 using Eq3.10
  Update predicted_Scores



Event : OnUpdate
Inputs: Users, CurrentUser, K
Outputs : SimilarUsers

For each user u' in Users
  Sim(u,u') 🡠 eq3.9

similarUsers = Find best K sim(u,U)
```

Algorithm 3.1: pseudo-code function of ItemSimilarity agent

3.2.4 “Predictor” Agent

The next agent is named Predictor, which is responsible for calculating the error and recording the predicted results. This agent only activates on the NewEntry event, and if the prediction records reach a specified number, it calculates the system's error regarding actual and predicted values. Based on the abilities of the system in use and the processing volume, this amount can be changed and adjusted. For example, in the simulations, after every 1,000 predictions made in the system, the error value of the specified values, that is, the predicted scores are recorded relative to the actual values, and ultimately, the error rate is calculated based on the average of the values calculated and then will be sent to the output. Another task for this agent is to create a new recommendation for each user that has been calculated in the previous section, and the middle results are stored in a file. This file can be collected at the end on a shared server or by a shared-memory sharing method to be available to other servers. This will make it easy to review and offer new items to new users anywhere in the world. Algorithm 3-2 contains an executable pseudo code of predictor agent.

```
Agent Predictor

Event : new_Entry
Inputs : Scores, Predicted_Scores
Outputs : Error

N = 0;
For each new score in Scores
  E = (Scores – predicted_Scores)^2;
  N = N + 1;

Error = sqrt(E/N);
```

Algorithm 3.2: pseudo-code function of Predictor agent

3.2.5 “Recommender” Agent

As discussed earlier, in any multi-agent Holonic system, we need a central observer or central agent as the supervisor. This is the role of the Recommender agent in the proposed model. This agent first receives incoming records including the ratings. With a simple preprocess, the agent will first categorize the rating, item number, and user number, and then it will activate the newEntry by manually calling the event. It should be noted that at the beginning of the running of the central Recommender agent, the two Prediction and ItemSimilarity agents will run to the required number (depending on the servers or data volume). Then by activating the new Entry event, both agents are automatically activated, because they both implement and override the event within themselves. The ItemSimilarity agent according to the new score, the user's number and the items, find the similar items for the user and sets each item in the list between 1 and 5 for the user (the values above 3 means that this item is suitable for a recommendation). The predictor agent is also activated by storing the new 1000 records each time and calculates the prediction error.

In the continuation of the activity, the Recommender agent invokes the onUpdate event by recording and predicting new ratings for each value n (number from 100 to 1000), then activates ItemSimilarity agent inside the event, and m represents the user similar to the current user. The ItemSimilarity agent's output which is the similar users will get back to Recommender and this agent will start to calculate recommended items again by re-calling the newEntry event for every similar user and finally updates the list of items. This way, if a user does not have a new activity, he will still have the chance to be updated when recording a few ratings. The main feature of the designed model is the simultaneous operation of intelligent agents, such that the Recommender agent can split a task into two other agents, and at the same time, other sample agents from the same type as this agent are made to speed up the process. The algorithm 3-3 is a pseudo-code implementation of the Recommendation agent.

```
Agent Recommender

Event : On_initialize
Inputs : Raw Data(D)
Outputs : Predicted_Scores

For each row r in D
  (Score,Item,User) 🡐 preprocess(r)

Call event newEntry
After each 1000 row in D
  Call event onUpdate

For similarUsers with current user
  Call event ItemSimilarity

Shut Down All remaining Agents
```

Algorithm 3.3: pseudo-code function of Recommender agent

3.3 Real-time Parallel Execution

The proposed method solves the two main problems of a recommender system in large-scale sites and databases. The first problem was the need to train the whole train set over time. For example, a system that uses matrix factorization to predict new ratings needs to be redefined for new items or new users that have no value to recalculate the user-product matrices. On the other hand, the use of a system that runs consecutively causes only one part of the calculation of the similarity of the product or the similarity of the user or the prediction of the new ratings and the calculation of the error run at any moment, while in the introduced Holonic model each section can run in parallel and without any dependency on the other sections. Along with this feature, the proposed system, unlike most of the other approaches that require a large number of records to start the prediction, can start the process of recommending from the first moment, even with two products and two users.

3.4 Continuous Updating Feature

Another important ability of the proposed approach is the continuous updating of the information and a list of proposed products. In the onUpdate event, users similar to the current user are selected, and their items are also updated. The main reason for designing this idea is that when a new user or a new rating is registered in the system, there is a possibility that the rated item may not exist in the list of similar users' list. Therefore, by choosing the users similar to the current user u, it will be possible to examine the interest rate and rating of the item in question (item i) according to the Cosine equation. Also, if a user has not been updated for a long time, and if the similar items and users have not recorded any ratings during this time, they will change the user's wish list and his recommended items.

3.5 More on the Holonic Structure

The ability to define more than 1 set of the intelligent agents for the Holonic (other than the Recommender agent), is another feature of the implemented model. It is not necessary for an agent to take responsibility for reviewing and calculating all the new ratings when a new score is entered into the system every second. Instead, some agents equal to the number of available computers, servers or even internal processors of a computer can be defined to speed up the calculation process. Since each agent is separate and independent, and the results are transferred to the supervisor at the same time, the calculations are performed separately, and of course, the results are stored in a shared file that can be shared.

The design of Holonic multi-event system may increase the speed, efficiency, and improvement of accuracy due to multiple updates, but there is still a challenge. Defining and designing different agents does not mean the parallel execution of all samples of an agent, and each agent ought to wait for agents of the same type due to resource limitations. There is a need for a parallelization method or assignment of processing resources to each agent. Without the design of a model with parallel implementation or assigning more than one resource to intelligent agents, the implementation of the program will be very slow. Suppose a case of processing Amazon data with 19 gigabytes, using a computer that can only use 8 gigabytes or 16 gigabytes of RAM each time, by folding the entire data into the hypothetical 20 MB sections and converting to about 1000 new datasets; it will probably require the system to run the procedure for thousands of times to complete the operation.

3.6 A Distributed Solution using Hadoop and map-reduce Technology

Using the Hadoop system and map-reduce technology will solve this problem. map-reduce technology is designed to solve problems with bulk datasets. Usually, a map-reduce model consists of two parts of the map and Reduce, and sometimes with an intermediate stage named Merge. The main task of the map-reduce framework is to provide a mechanism for data entry that includes massive information, instead of fully entering the main memory of the computer, which will also be accompanied by the filling and error of the RAM, to enter the main memory in separated sets of data. The framework provides the mechanism for dividing data and input files and data streams, and the encoder does not need to struggle with input methods or disunite them. The main challenge in using the map-reduce framework is the use of two main stages so that the problem-solving method can be linked in separate segments (Thusoo, et al., 2009).

A simple example may slightly simplify the explanation above. Suppose we have 200 million records and aim to find the maximum number in this record. Due to the inability to load all data into a single computer, and by rewriting the Map function in the map-reduce framework, the maximum value for each section of the dataset being received will be calculated. In the next step that refers to the Reduce function, all the values will be received. Suppose the system sends a million data to the Map function every time and therefore there will exist 20 maximum values. Now in the Reduce function, the remaining values will also be examined to find the maximum value, and the final value is the optimal maximum value of the dataset.

In addition to data segmentation, the Hadoop technology provided in the form of a framework, allows each section of the data to be run on the part of the resources (processor and RAM). For example, a computer with a 2-core processor can simultaneously execute two map functions and increase the speed of the program. Note that the Hadoop framework automatically does the allocation of resources and executing codes. In addition to allocating local resources, Hadoop can distribute data on the server resources, or even if we have some computers on a network, the data and computing operations can be distributed on each computer. This operation will increase the speed and efficiency with two constraints: first, if the map function for each run is independent, and secondly if the final result can be cumulated within the Reduce function.

The problem is not always this simple. For example, suppose there are 20 million sample data that has to be divided into 10 clusters, each of which has N features. If 20 runs take place with one million data per map, and each time there are 10 clusters, we will probably have 20 * 10 clusters with minimal common features because different samples are checked in each map function. In such a case, a solution should be designed that allows the final results to be uniformly integrated with the appropriate combination in the Reduce function.

The proposed method has been implemented on the map-reduce frameworks in Hadoop to increase the processing speed and efficiency, as well as the possibility of distributing the designed model. With this condition, we will be able to calculate each segment of the dataset individually and on a separate resource, and then the results of separate performances will be combined using the method developed in the Reduce function. In the map section, the designed recommender system model has been run on each of the dataset segments separately. Then based on the existing users, items, and the recorded ratings, the ratings for items with no scores will be predicted, and new scores will be recommended to them. The output of this section includes new predicted ratings for each user.

In the Reduce section, these results need to be combined. For instance, a user may receive n new products as a recommendation in a run, while it will receive m new items as another recommendation in a different run. The current user must receive m + n recommended items in the new version. On the other hand, there may be a commonality between these items, or an item that is being recommended in an implementation may receive a low rating in another run, in which case the ratings must be combined. Outputs of each running of the map function must be saved in a file on every run. When there are multiple ratings for one item or multiple ratings by one user, the average of the ratings needs to be calculated. Calculating the average in the integration section has solved this challenge. Thus, if conflicting and different values are recorded for a user-item, an average value will be recorded by calculating the average of the features of the item and interest of the user. The implemented model is fully represented in the flowchart shown in Figure 3.1. Figure 3.2 also shows the relationship between the agents designed in the Holonic.

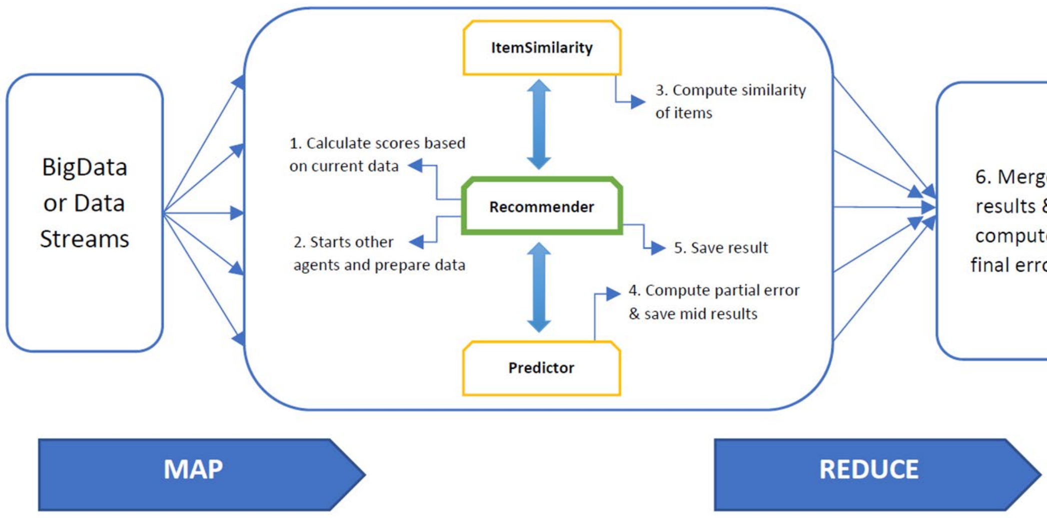

BigData or Data Streams
ItemSimilarity
3. Compute similarity of items
1. Calculate scores based on current data
Recommender
2. Starts other agents and prepare data
5. Save result
4. Compute partial error & save mid results
Predictor
6. Merge results & compute final error
MAP
REDUCE

Figure 3.1: Designed Holonic multi-event recommender system flowchart

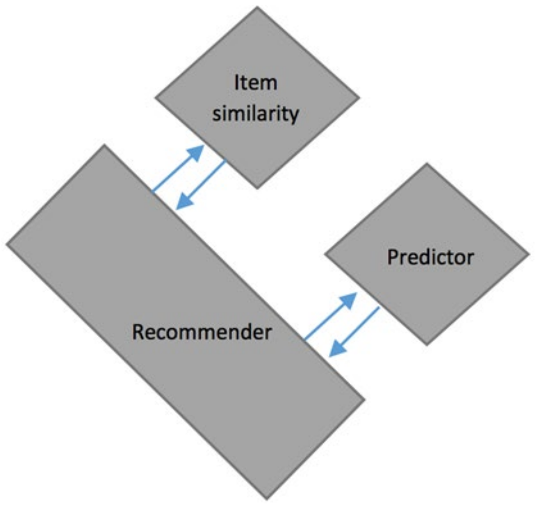


Figure 3.2: Schematic of the interaction of agents in the proposed Holonic model

3.7 Methodology Closure

In this section of the thesis, the proposed method and the details of the operation of the designed model were discussed. The implemented method uses the cosine equation introduced in the 2-28 relation in the calculation of item-item or user-user similarities.

The HMES has been used for a faster and separate calculation of the different segments of a large dataset or a data stream. This also contributes to a quicker update of the database (knowledge base) of the proposed system. On the other hand, this will cause a platform to be provided that each agent will try to predict ratings for other items that the user has not given any scores to, separately from other agents. By implementing the proposed model on the map-reduce Hadoop framework, the possibility that the program was designed to be distributed on local and distributed network resources has become possible. In the next chapter, various experiments are designed to evaluate the implemented method. Also, several similar and different models are simulated for reviewing and comparing the proposed method with the usual methods of the proposed system, the results of which are presented in the form of charts and tables in the next chapter.

By implementing the implemented model on the Hadoop map-reduce framework, it became possible to run the designed program on local resources and distributed on the network. In the next chapter, various experiments are designed to evaluate the proposed method. Also, various similar yet different models are simulated for reviewing and comparing the proposed method with the usual methods of a recommender system, the results of which are presented in the form of statistical charts and tables in the next chapter.

Chapter 4

EXPERIMENTS AND RESULTS

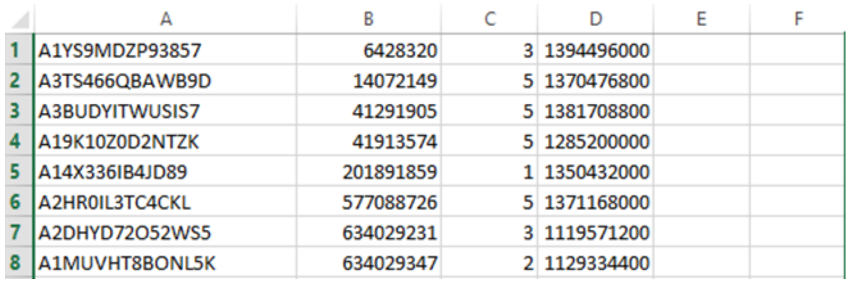
In the previous chapter, a new method of recommending an item to a user has been introduced. The main focus of the proposed method is to work on massive bulks of data or live data stream. That is why the implemented method uses successive learning during runtime instead of one-shot learning. In addition to that, the new system recommends users new suggestions based on the similarity of the items as soon as new information is received. By increasing the number of records, this operation will reduce the prediction error, but this will increase the processing speed and, of course, do not require initial calculations every time.

The use of several Holonic and Hadoop framework in the implemented method, have made it possible to distribute the computing load on several resources. In addition to that, since the intelligent agents are independent, one can perform several similar tasks to increase the procedure's speed by using the agents in a parallel timeline. Ultimately, the use of a central observer agent, occasionally called supervisor, will maximize the coordination between the designed agents and events. This chapter will examine the test conditions to analyze the results produced by the proposed model. Moreover, to compare the fairness of the implemented method with similar methods on the subject, another experiment has been designed, which is discussed in the following chapter. In the following sections, firstly the data used will be reviewed. Later on, the parameters and evaluation criteria of the experiment will be introduced and explained. Finally, the results of the tests will be analyzed, and a final conclusion of the analysis will be stated.

4.1 Dataset

To investigate and test a recommender system that is suitable for processing massive densities of data, one might safely say that no data resource is better than Amazon's public data sets. The full Amazon data is available to download at http://jmcauley.ucsd.edu/data/amazon/. From the "ratings only" category, a list of different items, including user ratings for different items can be downloaded. Generally, "ratings" files are .csv files which their values are separated by the comma mark (,). Each row of this dataset consists of a user ID, an item ID, a user's rating score to the product, and a system login time indicating the exact time which the rating has been recorded on the system. Figure 4.1 shows a sample section of a list called ratings_Musical_Instruments.csv. For example, in the first line, the user with the ID number A1YS9MDZP93857, has given a rating of 3 points to the item 6428320, and this record was recorded at the system time of 1394496000. The total files on Amazon, which include only recorded rating scores, are about 2.3 Gigabytes, including 83 million and 680,000 registered records (McAuley & Yang, 2016).

In the applied implementations, 80% of the dataset is used for training the train set, and 20% of it is used for testing. Because the system's performance is based on step-by-step learning and is not in a constant manner, available items for any given record for any user will be predicted. If the predicted items are from the test dataset, then the error based on the actual and predicted values will be calculated, and these similarity-recommendation calculations will continue until the end of the training set. Finally, a final calculation will be made for the error of the recommender system and test set. It should be noted that due to the need to compare with other methods, which are usually much slower than the proposed method, only 100,000 records were tested and calculated, and all data was not checked since other methods require several days to be calculated and recorded.



	A	B	C	D	E	F
1	A1YS9MDZP93857	6428320	3	1394496000		
2	A3TS466QBAWB9D	14072149	5	1370476800		
3	A3BUDYITWUSIS7	41291905	5	1381708800		
4	A19K10Z0D2NTZK	41913574	5	1285200000		
5	A14X336IB4JD89	201891859	1	1350432000		
6	A2HR0IL3TC4CKL	577088726	5	1371168000		
7	A2DHYD72O52WS5	634029231	3	1119571200		
8	A1MUVHT8BONL5K	634029347	2	1129334400		

Figure 4.1: A sample section of the Amazon data files in MS Excel

4.2 Programming languages, Settings and Parameters

To have a fair experiment, all simulations on the implemented model and comparable models have been done on a single laptop. The laptop has a 5-core processor with two physical cores, each capable of 2.5 GHz. The RAM used in the device has a storage capacity of 6 GB, and the SARL programming language has been used to implement the proposed model and the 2016 MATLAB for implementing the comparable models.

The implemented model has been coded using SARL, which is considered to be tailored for designing Holonic systems. A simplified model of all of the compared methods (MMD, SVD, Asymmetric, GA matrix factorization, ALS, Gradient descend, and Cosine similarity) have been implemented and used using Matlab. The pseudo-codes of these Matlab implementations are given in the appendix of this thesis.

SARL programming language is a language specific to the programming of multi-agent and Holonic systems. The language, which is part of the Janus 2.0 project, is

designed in Java and is presented in the Ellipse IDE. The following link is provided to download the directly embedded version of eclipse:
http://www.sarl.io/download/index.html. SARL is very similar to Java or the mother language C with some small differences and limitations in coding, which makes learning and using of this language very easy. This programming language enables the user to define events and agents, and automatically manages the events and synchronization of the operation. All events required by an agent such as start-up and termination have been designed in SARL, and if necessary, the user can add personal events to the set as well (Rodriguez, Gaud, & Galland, 2014).

The parameters included in the runtime and the tests are listed in Table 4-1. Changing these parameters has a serious influence on the speed and accuracy of the implementation of the program and the quality of the performance of the recommender system. The reason of this impact is that the defined values will determine the time of saving and merging the results, and by varying these values, the effect of the sparseness of the matrix will increase or decrease, resulting a direct effect in averaging; as the final average is the predicted number of the recommender system, the changes in this value will also be effective on the prediction error value. Other values, such as the number of similar users, will reduce or increase the sensitivity of updating.

At least 1000 records in the implemented simulations are required for the system to activate the similar users and item rating update phases. In other words, before the entry of 1000 records, the designed system will not enter the similar user phase. In simulating much larger data, of course, this number needs to be much larger, say, ten thousand or a hundred thousand because at the beginning of the process, users are different and the procedure of updating users and similar items will only slow down the execution of the program. After activating the similar user update phase, with the arrival of every 100 new rating records, ten similar users will be found, and their list of items will be updated. As the number of records that are to be used for updating increases, the speed of the program will be increased. However, the updating process itself will be slower. Also, increasing the number of similar users from 10 to more numbers will increase the variety, but will be accompanied by a slowdown in the implementation of the program.

Table 4.1: Parameters used in the simulations

Minimum number of records to start updating the similar users	1000
Number of records to repeat updates	100
The number of similar users per update	10
Minimum number of records to save the results of each Hadoop run	5000
The number of records to save the results of each Hadoop run	1000

As mentioned in the previous chapter, each Hadoop run must have intermediate results saved and integrate the middle results at the end or during the execution of the program. An intermediate file of the predicted results from the current Hadoop run will be saved after reviewing at least the first 5,000 records. There may be only one Hadoop run on a system at any time, or there may be a run on each core of the system, and have several Hadoop runs in parallel, or even have many runs on different servers to speed up the process. Using the shared file, it is easy to integrate the results so that all servers can use up-to-date information. For this reason, after 5,000 runs, the new prediction data is stored in the file, and after that, by every 1,000 new records that enter the system, each Hadoop run will save the new data. In this way, this information, along with other data from the parallel / serial Hadoop runs, should eventually be merged by averaging the calculated prediction results. With the increase of 5,000 and 1,000, the results will have less fluctuation because they will include more changes to records, but obviously, the process of updating will be slower.

All values listed in Table 4-1 are values that are based on the experience of different implementations, these values are set to not increase the error as much as possible, but also increase the speed of the program execution. By increasing the size of the data, changing the speed of the system on which the model will run on, or the degree of sparseness (the ratio of user ratings to the entire item-user matrix), these parameters should be re-adjusted and tested according to that system.

4.3 Evaluation Criteria

To evaluate the implemented method and compare it with other conventional methods, it is necessary to introduce the same evaluation criteria for all and to examine the result on its basis. In this section, the usual evaluation methods for the comparison of the proposed system will be introduced. These evaluation criteria apply to all simulations performed for the proposed model and other comparative methods, and the results and their comparison are available in detail in the next section.

4.3.1 Perfect Hit (PHIT)

As the first criteria of evaluation, the "perfect hit" or "PHIT" which is a common tool of comparison is used. In this method, the estimation of accurate predictions is calculated. The reason for using this method is to assess the effectiveness of the proposed method in the real world. In each recommender system, the output of the process is integer values with decimal pointed accuracy. For example, a prediction may be 2.5 or 2.03 or 2.98, while it is known that the real values for each registered rating are natural numbers that do not have decimals. For this reason, the calculated value of 2.98 must be changed into 3, and then the error is calculated with the real value. In addition to that, another operation that occurs in the calculation of PHIT is that the value one is considered in the number of perfectly correct predictions and the value of 0 is considered for each false prediction. In this case, the total of correct predictions divided by the total predictions of the test set will determine the correct detection percentage. This assessment method is a good way to determine the quality of a recommender system for real systems because in the real world it is necessary that the predicted values are as close as possible to real values and the predictions can be used. Equation 4-1 shows how to calculate PHIT. Round means rounding the number to the nearest natural number and also P is the predicted value that is compared with the actual value of R. Finally, if the values are equal, the difference between the two numbers is 0, and as a result a number is added to the sum, otherwise no value will be added to the sum (Ronen, Koenigstein, Ziklik, & Nice, 2013) (Bobadilla, Serradilla, & Hernando, Collaborative filtering adapted to recommender systems of e-learning, 2009).

$$PHIT = \frac{\sum_{u,i} round\left(P_{u,i} - R_{u,i}\right) == 0}{N} \tag{4-1}$$

4.3.2 Mean Percentage Rank

Another evaluation method named Mean Percentage Rank exists that is used to determine the quality of the recommender system. This method calculates the proportion of correct predictions alongside half of the correct predictions as the total value. The Mean Percentage Rank benchmark attempts to determine the effect of correct predictions by reducing the impact of false predictions. Therefore, the small differences between the two proposed systems mean that the two systems are operating very similar to each other and the value of the better model's work will be further determined by increasing this value. In equation 4-2 the way of calculating the percentage rank is shown. The value of CF, which is short for correct frequency, determines the number of correct predictions and the F signifier means false predictions. The final total will be calculated by dividing to the total predictions. The percentage rank will be calculated for each number. For example, if there are 5 rating values, a rank percentage will be calculated for each rating, and finally, the total average will determine the Mean Percentage Rank (Herlocker, Konstan, Terveen, & Riedl, 2004).

$$mean\ percentage\ rank = \frac{CF + 0.5F}{N} \tag{4-2}$$

In the next section of this chapter, the evaluation methods and quality of each model will be reviewed. In addition to the evaluation methods introduced, the simulation runtime for each model will also be introduced as the fifth benchmark. No matter how good a method is regarding quality, the runtime is very important. Increasing precision as much as a few percentage points, if accompanied by a lower speed and a longer duration of the program as big as several hours or days, may not be a good solution. Therefore, it is very important to consider the time factor in a recommender system that deals with massive amounts of data and data streams.

4.4 Experiments and Results

The next evaluation method that has been compared is the PHIT benchmark, which focuses on the correct prediction ratio. This figure is plotted for all four models of the previous figure in Figure 4.2. The PHIT method shows how many fractions of the predictions have been made completely. Given the following figure, it is clear that all three methods are typically above 97% in the correct predictions, and the difference between the three models in the true prediction is negligible. To better understand the superiority of the methods on each other, the displayed range is 5% from 95% to 100%.

The implemented method is 2% better than the best method available in the following figure.

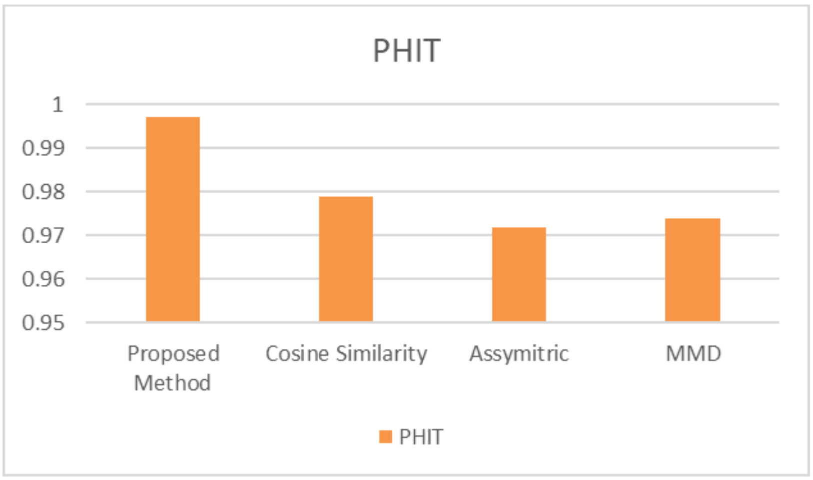


Figure 4.2: Comparing the PHIT assessment method for all introduced methods

All three of the compared methods with more than 97 percent have shown a high prediction accuracy and only differ in the third digit, indicating that they have almost the same ability and, of course, the ability to accurately predict the test set. Perhaps using a combination method with the presence of three methods or two methods of the three methods as the base model in the suggested HMES recommender system can relatively improve the final model, which is presented in the next chapter.

The Mean Percentage Rank, as already mentioned, is an appropriate evaluation method to determine the error rate in the prediction along with the correct predictions. Figure 4.3 shows the MPR behavior for each of the four introduced models. Depending on how much MPR value is closer to the maximum rating score limit (5 at Amazon dataset), the number of true predictions is much higher than the false predictions. The implemented method has a better performance in MPR, and compared with other comparable methods, the proportion of error is less for each rating. When the MPR is high, and the prediction error is low, it means that the ratings in the data are distributed uniformly, and the correct field predictions are not specific to one particular score, and the system can predict scores from 1 to 5 successfully. Sometimes, in some recommender systems, the average behavior of 4-2 is easier to predict, and systems can predict values of 4-2 in an over-fit manner. And a weakness of this aspect can be easily detected by the MPR evaluation method (Bobadilla, Serradilla, & Bernal, A new collaborative filtering metric that improves the behavior of recommender systems, 2010).

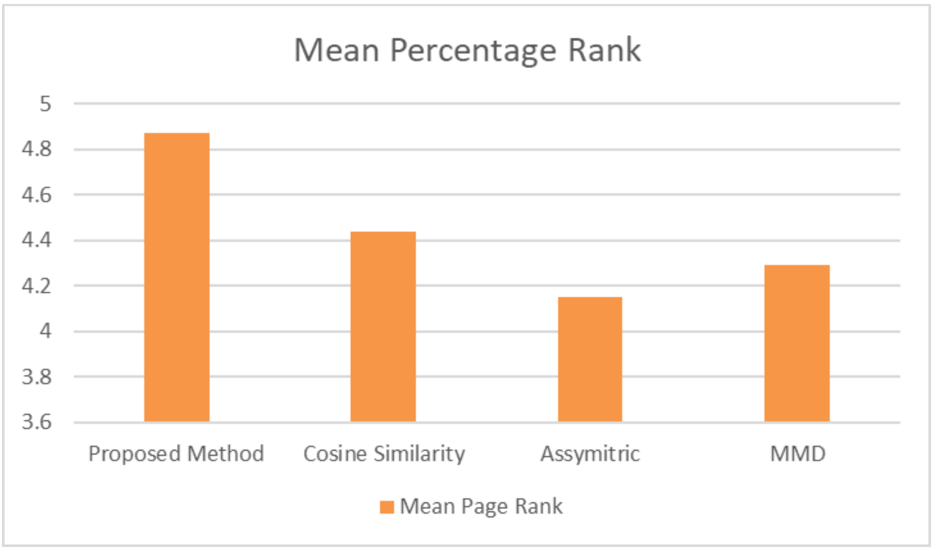


Figure 4.3: Comparing the MPR assessment method for all introduced methods

After reviewing all the evaluation methods, it is the time to compare the execution time of all the introduced models. No matter how precise the predictions are or how low the error values, a method has to get a solution within a limited time, and if the calculation process takes too long, then the method is not suitable for a recommender system that is supposed to work on big data or live data streams. Figure 4.4 shows the comparison of the time of each of the eight models. The proposed method examines all considered records in less than a minute. It should be noted that only 100,000 records have been tested in the experiments, and the reason for that is that methods such as neighboring methods and matrix factorization are too slow. The suggested method required only 52 seconds to complete the entire work, and the closest model in the matter of runtime are SVD or LS with about 7 minutes of runtime.

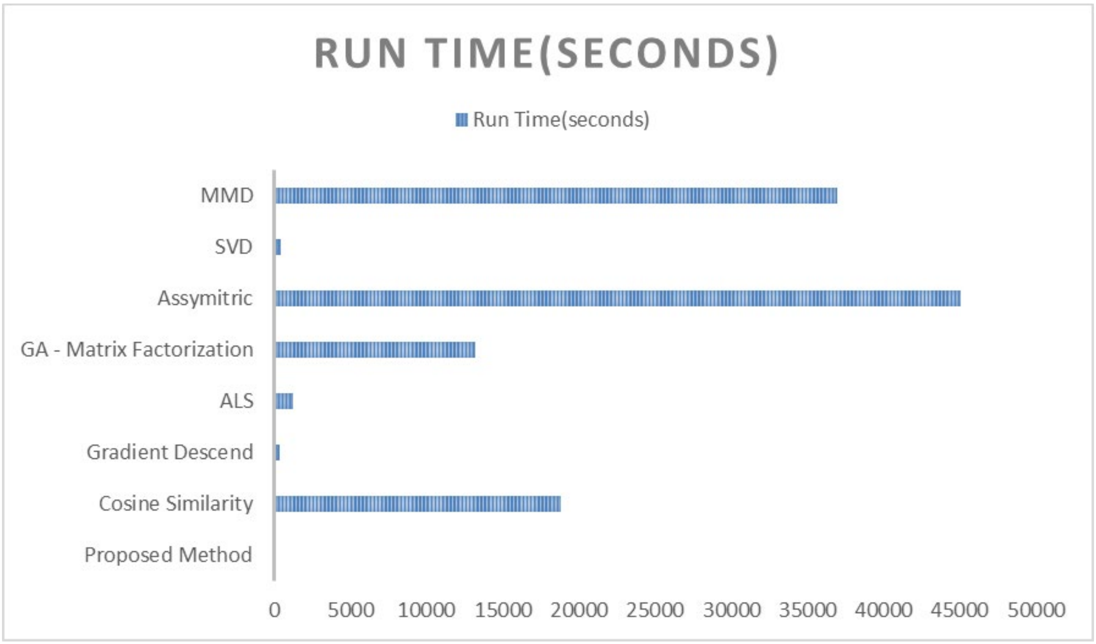


Figure 4.4: Runtime comparison for all introduced methods

The matrix factorization method with genetic algorithm has spent a lot of time, which is about 15,000 seconds, due to the need for this evolutionary algorithm to repeat itself. Neighboring methods all have very low execution speeds, which is the main reason for it is the serial operation of the calculations. In neighboring methods, computational equations should be rented for each pair of users, and if there are 1,000 users, then 21,000 repeats of the loop are required to complete the entire algorithm. Under these circumstances, it seems that the use of neighboring methods, even though they are highly precise, cannot be used in huge recommender systems because of the high volume of equations. In contrast, matrix factorization techniques such as SVD and descending gradients are getting to a result more quickly because of a higher speed and despite a bigger error, which they have. In this situation, the proposed method seems to be an extremely fast recommender system that combines precision and fast execution. It may be possible to combine the matrix degradation and implemented HMES systems together and create a recommender system with higher precision and also increase the precision concerning state-of-the-art solutions.

4.5 Evaluations Summary

Experiments conducted on Amazon's data set show the superiority of the suggested method compared to other comparative approaches. The methods that are being compared are all the usual and popular methods that today have many admirer fans. Of course, all of the above-mentioned methods have come up with a faster computing process by parallelization or computing-reduced models, although they will not be comparable to the proposed model due to the one-stage learning/training behavior. The implemented recommender system, in addition to reducing the computational time, also greatly improves the accuracy of the predicting of ratings, which makes this a suitable model for implementation on massive data sets. On the other hand, the use of the Hadoop has helped the model also have the ability to implement segmentation based recommending computations.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

Recommender systems, considering their users' information, are trying to improve their behavior to increase sales, attract more customers, and correct their problems. A recommender system can be used on social networks to increase the positive feedback rate; sometimes huge Internet shops use these systems to increase the purchases, as well as some service providing websites, which use these systems to get a better understanding of how users are interested in their services. Different methods have been designed and introduced to improve the behavior of recommender systems, and each has been successful in improving the system based on one or more mathematical and artificial intelligence models. Although none of the methods are the same and each method has a unique behavioral system, all of these models follow the same goal, which is increasing the benefits of using recommender systems. The basis of these systems in all cases is based on the item (product, service or information) and the user.

In a sales website, or a movie rental service, or even social networks, if the item is considered to be a product, a movie, or a related advertisement; everything is divided into two domain and user categories that have tight links to each other. Of course, if there were explicit information about what opinion each of the users had about the items of a website, without the use of any third-party system, some suitable recommendations could have been provided for the users. Since providing a questionnaire for all items or asking users to comment on all items is impossible and inaccessible, methods that make the best estimation about the interests of users are needed to be used.

Some methods based on the similarity of items and others based on the similarity of users, decide to predict and provide an estimation of the interest of users in items. Other models have also been developed to make recommendations by combining the user-user and item-item similarities to each other. Among these, one of the most famous and most successful ways is the cosine similarity method (Ricci et. al., 2010). The cosine similarity method, based on the current user ratings and opinions of the users, makes it possible to estimate the interest of users in other items.

The reason for this method's popularity is the simplicity of the implementation and availability of the information required by this system. Although the user ratings of a website about its items are presented in multi-million matrices, it should not be overlooked that this information is always available in the system and accessing to them is possible with only a few simple queries in the database. Unlike this model, in methods that work based on the similarity of items or even users to each other, not only that the calculated are not 100% reliable, but there are different standards for similarity, which makes other methods much more complex than the matrix factorization approach.

Holonic systems are used in the implemented method with the goal that each agent takes on some of the tasks and, of course, intelligent agents can work together in parallel. In the Holonic recommender system, considering an agent as the supervisor, several versions of a particular agent can be produced if needed. In this way, in addition to improving the quality, calculation speed will also greatly increase. Two agents will operate under the supervisor's control. The first one's duty is to calculate the similarity of the items to each other and to assign the product's rating to the users. The other factor has the task of calculating the error of each user's proposed items and storing intermediate data.

Item Similarity agent produces items list for each user based on the Cosine Similarity of both users and items. This agent then finds item intersections for those users who are similarly based on the Cosine Similarity. Cosine Similarity is chosen as the core similarity basis of the suggested method since it is simple to implement and probably lower complexity of the model. It is decided to benefit from map-reduce technology of Hadoop framework in order to make the system quicker. Using map-reduce will help the model to process big-data on divided smaller parts, which enables the system to distribute each part into server clusters, simple computers or even the processors of a computer.

In the conducted experiments and analysis in Chapter 4, and comparing the results of the implemented method with other methods, the proposed method can be able to reduce the prediction error, and in this case, undoubtedly, the suggested approach was the best method studied among all models. According to the test results of the compared models in PHIT and MPR assessments, the closest models to the proposed method were the cosine similarity model and the asymmetric model. The reason why the implemented method works better than the simple cosine similarity model is to update the recommendations of each user by increasing the size of the data. The cosine method will calculate the similarity for each user once, whereas the proposed method will repeat the calculation with the frequency of the number of Hadoop runs or the number of intermediate updates of the operation.

In the discussion of time comparison, the implemented method is not comparable with the introduced serial operating models due to its parallel execution capabilities and also due to the fact that there is no need for a serial review of the program. This runtime will be surely reduced if implemented on server hardware, which is another strong side of the implemented model.

5.2 Future Work

In the recent years, a lot of studies have been dedicated to recommender systems, and now researchers are trying to improve the existing methods. Hence new methods are introduced and presented annually. In this section of the paper, there are some suggested ideas that can be applied to the proposed model and the context of multi-agent systems in order to improve the approach and increase the possibility of its implementation.

The first approach that is suggested as future work is to compare the speed of the designed model based on running on a cluster with the same conditions with other methods. Large social media or massive online marketplaces all have very powerful systems and servers to process their information, and one of their tools is the mentioned cluster system. If the suggested model can be implemented in parallel with a cluster, then a more accurate comparison can be made of the possibility of implementing the proposed method in the real world.

Various methods have been introduced to improve the quality of the work of the recommender system so far, as in this study seven known methods have been introduced and compared. The main strength of the implemented method is the use of Holonic, and then provides a mechanism to improve performance on one or more systems, and this same strength point can be used with other similar methods. For example, parts of the main user-product matrix or rating matrix can be divided into smaller matrices, and then apply matrix analysis methods in parallel by Hadoop and by running different agents on each matrix. Methods such as the Asymmetric Cosine model or other similarity models can be used instead of Cosine Similarity, and these methods are expected to have the same or close accuracy to the proposed method and may even improve it.

One of the main weaknesses of academic models and of course the implemented method is that the feedback from the users on a recommendation after they receive it is not influential. When the system gives a user a few new recommendations, if the user visits those offers, which will be a positive feedback for the system. Otherwise, the specified offer will, in fact, be a mistake or failure for the system. Taking it to a higher level, a customer's purchase of a previously recommended item would be a bonus score to a recommendation. If the user is convinced to purchase a recommended item, it definitely is a big step forward for the system. It is suggested to innovate a mechanism where the Predictor agent that is already in charge of performing the error calculation, determine the feedback and then, based on machine learning or by changing other parameters, improve the prediction status during the execution of the program, and stop a permanent, ineffective or low-valued process from running further.

REFERENCES

Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *Knowledge and Data Engineering, IEEE Transactions on.*, 734-749.

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook.* Springer Publishing Company.

Bellogin, A., Cantador, I., Diez, F., Castells, P., & Chavarriaga, E. (2013). An emprical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Trans. Intell. Syst. Technol.*, 1-29.

Bobadilla, J., Serradilla, F., & Bernal, J. (2010). A new collaborative filtering metric that improves the behavior of recommender systems. *Knowledge-Based Systems, 23*(6), 520-528.

Bobadilla, J., Serradilla, F., & Hernando, A. (2009). Collaborative filtering adapted to recommender systems of e-learning. *Knowledge-Based Systems*, 261-265.

Burke, R. (2000). Knowledge-based recommender systems. *Encyclopedia of library and information systems*, 175-186.

Burke, R. (2010). Evaluating the dynamic properties of recommendation algorithms. *RecSys '10 Proceedings of the fourth ACM conference on Recommender systems*, 225-228.

Gaud, N., Gechter, F., Galland, S., & Koukam, A. (2007). Holonic multiagent multilevel simulation application to real-time pedestrians simulation in urban environment. *IJCAI'07 Proceedings of the 20th International joint conference on Artificial intelligence* (pp. 1275-1280). Hyderabad: Morgan Kaufmann Publishers Inc.

Gemulla, R., Nijkamp, E., Haas, P. J., & Sismanis, Y. (2011). Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent. *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 69-77). ACM.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS), 22*(1), 5-53.

Hoek, W. V., & Wooldridge, M. (2008). Multi-Agent Systems. *Foundations of Artificial Intelligence*, 887-928.

Horvath, T. (2012). Recommender systems. *Tutorial at the conference at Znalosti 2012.* Mikulov: Pavol Jozef Safarik University.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer, 42*(8), 30-37.

Lathia, N., Hailes, S., Capra, L., & Amatriair, X. (2010). Temporal diversity in recommender systems. *SIGIR '10 Proceedings of the 33rd international ACM SIGIR conference on research and development in information retrieval*, 210-217.

Lops, P., Gemmis, M. d., & Semeraro, G. (2011). Content based recommender systems: state of the art and trends. In *Recommender systems handbook* (pp. 73-105).

Mahara, T. (2016). A new similarity measure based on mean measure of divergence for collaborative filtering in sparse environment. *Procedia Computer Schience*, 450-456.

McAuley, J., & Yang, A. (2016). Addressing complex and subjective product-related queries with customer reviews. *Proceedings of the 25th International Conference on World Wide Web* (pp. 625-635). International World Wide Web Conferences Steering Committee.

Parambath, S. A. (2013). *Matrix Factorization Methods for Recommender Systems.* Umea University.

Phelan, O., McCarthy, K., & Smyth, B. (2009). Using Twitter to recommend Real-time tropical news. *Proceedings of the third ACM conference on Recommender systems* (pp. 385-388). New York: ACM.

Pirasteh, P., Hwang, D., & Jung, J. J. (2015). Exploiting matrix factorization to asymmetric user similarities in recommendation systems. *Knowledge-Based Systems, 83*, 51-57.

Resnick, P., & Varian, H. R. (1997). Recommender Systems. *Communications of the ACM*, 56-58.

Ricci, F., Rokach, R., & Shapira, B. (2010). *Recommender Systems Handbook.* Springer Publishing Company.

Rodriguez, S., Gaud, N., & Galland, S. (2014). SARL: a general-purpose agent-oriented programming language. *2014 IEEE/WIC/ACM International Conference on Intelligent Agent Technology.* Warsaw: IEEE Computer Society Press.

Ronen, R., Koenigstein, N., Ziklik, E., & Nice, N. (2013). Selecting content-based features for collaborative filtering reccommenders. *Proceedings of the 7th ACM conference on Recommender systems, RecSys '13* (pp. 407-410). New York: ACM.

Salakhutdinov., R., & Mnih, A. (2007). Probabilistic Matrix Factorization. *NIPS*, 605-614.

Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 263-278.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). *Application of Dimensionality Reduction in Recommender System - A Case Study.* Minneapolis: University of Minnesota - Army HPC Research Center.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web WWW '01* (pp. 285-295). Hong Kong: ACM.

Schafer, J. B., Frankowsk, D., Herlocker, J., & Sen, S. (2007). Collaborative Filtering Recommender Systems. *The adaptive web*, 291-324.

Shani, G., & Gunawardana, A. (2015). Evaluating Recommendation Systems. In *Recommender Systems Handbook* (pp. 265-308). Springer.

Shardanand, U., & Maes, P. (1995). Social information filtering: algorithms for automating word of mouth. *International Conference on Computer-Human Interactions.* New Work: ACM Press/Addison-Wesley Publishing Co.

Srinivas, M., & Patnaik, L. .. (1994). Genetic Algorithms: A survey. *Computer, 27*(6), 17-26.

Venkatraman, S., Fajd, K., Kaspi, S., & Venkatraman, R. (2016). SQL Versus NoSQL Movement with Big Data Analytics. *Int. J. Inform. Technol. Comput. Sci.*, 59-66.

Wu, D., Zhang, G., & Lu, J. (2015). A fuzzy preference tree-based recommender system for personalized business-to-business e-services. *IEEE Trans. Fuzzy Syst.*, 29-43.

Zanker, M., Felfernig, A., & Friedrich, G. (2011). *Recommender systems: An Introduction.* New Tork: Cambridge University Press.

APPENDICES

Appendix A: Pseudo-code of MMD used in the comparisons of chapter 4

```
MMD Pseudo-code
Inputs : Rating Matrix(R)
output : Predicted Rating Matrix (R')
-------------------------------------
for each user U row in R
      find Items Users U' such that U(items) Intersects with U'(items)
      Isize <- number of U items
      I'Size <- number of U' items
      MMD  <-  1/(average  of  ratings  of  ISize-I'Size)^2  - (1/Isize - 1/I'Size)
      CS(U,U')  <-  Cosine  Similarity  of  U  and  U'  according intersected Items I
      JAC(U,U') <- Isize / |all items|
      SIM(U,U') <- JAC(U,U') * CS(U,U')*MMD(U,U')

for each element r in Rating
      avg_r <- average of r ratings
      U' <- select K similar users based on CS
      R'(r) <- SIM(U,U')*avg_r/sum of SIM(U,U')
```

Appendix B: Pseudo-code of Asymmetric Similarity used in the comparisons of chapter 4

```
Asymmetric Similarity Pseudo-code
Inputs : Rating Matrix(R)
output : Predicted Rating Matrix (R')
-------------------------------------
for each user U row in R
      find  Items  I  in  which  Users  U'  such  that  U(items)
Intersects with U'(items)
      AS(U,U') <- sum(R(U,I)) - sum(R(U',I)) * |I|/ all_items
of U and U'

for each element r in Rating
      avg_r <- average of r ratings
      U' <- select K similar users based on CS
      R'(r) <- AS(U,U')*avg_r/sum of AS(U,U')
```

Appendix C: Pseudo-code of matrix factorization method using Genetic Algorithm used in the comparisons of chapter 4

```
GA Pseudo-code
Inputs : Rating Matrix(R) , dim dimension of hidden variables,
cross-over rate CR, mutation rate MR, population psize
output : two matrix P,Q that could produce R' as predicted
matrix
------------------------------------
P <- size of matrix as rows(R)*dim
Q <- size of square matrix as dim*cols(R)
initiate pop for P,Q randomly as Psize
loop until stopping criterion (max iteration)
      make newPop Psize*CR offspring using pop
      mutate newPop based on MR
      allPops <- [pop,newPop]
      for each element in allPops
            E(i) <-fitness_evalutioan :
                  sum(|R-P*Q|)
      select best Psize among allPops
```

Appendix D: Pseudo-code of Descent Gradient used in the comparisons of chapter 4

```
Gradient Pseudo-code
Inputs : Rating Matrix(R) , dim dimension of hidden variables,
alpha constant rate, beta constant rate
output : two matrix P,Q that could produce R' as predicted
matrix
------------------------------------
P <- initiate matrix as rows(R)*dim randomly
Q <- initiate square matrix as dim*cols(R) randomly
loop until stopping criterion (max iteration)
     e <- sum(|R - P*Q|)
     t <- alpha*(e*Q - beta*e*P)
     P <- P + alpha(e*Q - beta*P)
     Q <- Q + alpha(e*P - beta*Q)
     change alpha,beta towards minimizing |e|
```

Appendix E: Pseudo-code of Cosine Similarity method used in the comparisons of chapter 4

```
Cosine Similarity Pseudo-code
Inputs : Rating Matrix(R)
output : Predicted Rating Matrix (R')
------------------------------------
for each user U row in R
    find Items Users U' such that U(items) Intersects with U'(items)
    CS(U,U') <- Cosine Similarity of U and U' according intersected Items I

for each element r in Rating
    avg_r <- average of r ratings
    U' <- select K similar users based on CS
    R'(r) <- CS(U,U')*avg_r/sum of CS(U,U')
```

Appendix F: Pseudo-code of Total Average method used in the comparisons of chapter 4

```
Total Average Pseudo-code
Inputs : Rating Matrix(R) , dim dimension of hidden variables,
alpha constant rate, beta constant rate
output : two matrix P,Q that could produce R' as predicted
matrix
-------------------------------------
P <- initiate matrix as rows(R)*dim randomly
Q <- initiate square matrix as dim*cols(R) randomly
loop until stopping criterion (max iteration)
     for each user U
          IU <- items of user U
          vector <- IU*(P*Q)
          matrix <- vec*vec' *|U|*alpha
          x <- matrix/vector
          updates P,Q based on minimizing |P(U,dim)*Q(dim,U)
- R(U,all_items)|
```