THE REPUBLIC OF TURKEY

BAHCESEHIR UNIVERSITY

BUILDING A HYBRID RECOMMENDER SYSTEM USING APACHE SPARK

Master's Thesis

MUSTAFA FATİH ÇETİN

İSTANBUL, 2019

THE REPUBLIC OF TURKEY

BAHCESEHIR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE

BIG DATA ANALYTICS AND MANAGEMENT

BUILDING A HYBRID RECOMMENDER SYSTEM USING APACHE SPARK

Master's Thesis

MUSTAFA FATİH ÇETİN

Supervisor: ASSIST. PROF. DR. SERKAN AYVAZ

İSTANBUL, 2019

THE REPUBLIC OF TURKEY

BAHCESEHIR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
BIG DATA ANALYTICS AND MANAGEMENT

Name of the thesis: Building a Hybrid Recommender System Using Apache Spark

Name/Last Name of the Student: Mustafa Fatih ÇETİN

Date of the Defense of Thesis: 05.08.2019

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assist. Prof. Dr. Yücel Batu SALMAN
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Serkan AYVAZ
Program Coordinator

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Comittee Members	Signature
Thesis Supervisor Assist. Prof. Dr. Serkan AYVAZ	----------
Member Assist. Prof. Dr. Tarkan AYDIN	----------
Member Assist. Prof. Dr. Atınç YILMAZ	----------

DEDICATION

I would like to dedicate this thesis study to my wife, Yasemin, who has been very supportive during all phases of my study and has always urged me to finish what I'd started. Without her support and motivating behavior, it would be impossible for me to finish my thesis.

I would also dedicate this thesis to my sons Can Batur and Ahmet Eralp, who have given me the energy to finish this thesis during a very busy period.

I hope they will all forgive me for the time that I had to neglect them to finish my Master's program.

ACKNOWLEDGMENT

I would like to express my gratitude to my thesis advisor, Assist. Prof. Serkan AYVAZ, whose precious support and patience helped me throughout the thesis study. He has always motivated me and I have always felt his support and guidance during my Master's program.

I would also like to provide my appreciation to the committee members who were generous with their expertise and valuable time.

İstanbul, 2019 Mustafa Fatih ÇETİN

ABSTRACT

BUILDING A HYBRID RECOMMENDER SYSTEM USING APACHE SPARK

Mustafa Fatih ÇETİN

Big Data Analytics and Management

Thesis Supervisor: Assist. Prof. Dr. Serkan AYVAZ

August 2019, 53 Pages

The rapid advance of digital business after the millennium has exponentially boosted the options of available items to the users. Customized lists of recommendations assist people by limiting the number of options to a very few items that are related to the specific user and thus supporting them to have the final decision with ease.

The project intends to create a recommender system using a hybrid method. The two main algorithms used in the hybrid method are Alternating Least Squares and Negative Similarity Collaborative Filtering. Alternating Least Squares is a type of algorithm for matrix factorization, which can be parallelized during runtime. It performs relatively well when processing large scale datasets. Negative Similarity Collaborative Filtering, NSCF, technique tries to exploit the tastes of people at the opposite poles. The basic logic in this technique is the idea that one of the two people on opposite poles will love something that the other does not. The cascade hybridization method was used in the project to hybridize the algorithms and to create a better recommendation.

In the study, experiments were performed in a cascading way on the algorithms that are using MovieLens dataset, a well-known and widely used dataset in recommender system researches.

One of the well-known big data processing framework, Apache Spark, was used in the study. It provides the benefit of processing iterative algorithms in an efficient way and requires minimal processing time in comparison to map-reduce technique. It was observed that the usage of Apache Spark provided a faster way to perform iterative algorithms.

Keywords: Hybrid Recommender, Spark, Collaborative Filtering

ÖZET

APACHE SPARK KULLANARAK HİBRİT BİR ÖNERİ SİSTEMİ OLUŞTURULMASI

Mustafa Fatih ÇETİN

Büyük Veri Analitiği ve Yönetimi

Tez Danışmanı: Dr. Öğr. Üyesi Serkan AYVAZ

Ağustos 2019, 53 Sayfa

Özellikle milenyumdan sonra dijital ticaretin hızla yaygınlaşması, insanların kullanımı için sağlanan ürünlerin çeşitliliğini ve miktarını katlanarak arttırmıştır. Kişiselleştirilmiş öneri listeleri; insanlara ilgi alanlarına göre sunulacak seçenekleri en uygun şekilde daraltarak sunmaya yardımcı oldukları için, insanların son kararı vermedeki işlerini kolaylaştırmaktadır.

Bu projede hibrit yöntem kullanılarak bir öneri sistemi oluşturulmaya çalışılmıştır. Hibritleştirmek için kullanılan iki temel yöntem *Değişken En Küçük Kareler* yöntemi ile *Negatif Benzerlikli İşbirlikçi Filtreleme* yöntemidir. Değişken En Küçük Kareler algoritması, paralel bir şekilde çalışan bir matris çarpanlara ayırma algoritmasıdır ve büyük ölçekli veri kümelerini işlemede iyi performans göstermektedir. Negatif Benzerlikli İşbirlikçi Filtreleme tekniği, zıt kutuplar olarak tabir edilen insanların zevklerindeki farklılıkları kullanmaya çalışmaktadır. Bu teknikteki temel mantık, zıt kutuplardaki iki insandan birisinin sevmediği bir şeyi diğerinin seveceği düşüncesidir. Projede hibritleme yöntemi olarak kademeli hibritleme yöntemi kullanılmıştır.

Bu tez çalışmasında; öneri sistemleri çalışmalarında oldukça bilinen ve yaygın bir şekilde kullanılan MovieLens veri seti kullanılmıştır. Kademeli hibritleme yönteminde kullanılan algoritmaların birincisinin sonuçları, ikinci algoritmada da faydalanılarak en uygun sonuç elde edilmeye çalışılmıştır.

Bu projede; yaygın ve popüler bir şekilde büyük veri işleme sistemi olarak kullanılmakta olan Apache Spark altyapısı kullanılmıştır. Apache Spark, yinelemeli algoritmaların verimli bir şekilde işlemden geçmesini sağlar ve Map-Reduce tekniğine kıyasla en az işlem süresi gerektirmektedir. Apache Spark kullanımının, yineleyen algoritmaların işlenmesinde hızlı bir şekilde çalıştığı gözlemlenmiştir.

Anahtar Kelimeler: Hibrit Öneri, Spark, İşbirlikçi filtreleme

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

ALS	:	Alternating Least Squares
API	:	Application Programming Interface
CBF	:	Content-based Filtering
CF	:	Collaborative Filtering
FN	:	False Negative
FP	:	False Positive
kNN	:	k-Nearest Neighbors
MAE	:	Mean Absolute Error
NSCF	:	Negative Similarity Collaborative Filtering
PCA	:	Principal Component Analysis
PCC	:	Pearson Correlation Coefficient
PRC	:	Precision-Recall Curve
RDD	:	Resilient Distributed Datasets
RMSE	:	Root Mean Square Error
ROC	:	Receiver Operating Characteristics
SQL	:	Structured Query Language
SVD	:	Single Value Decomposition
TP	:	True Positive

1. INTRODUCTION

The World Wide Web and its usage by people have boomed at the beginning of the 21st century. The availability of accessing to internet through devices such as smartphones, tablets, computers, etc., have led people to access and generate data more easily and quickly. This also resulted in the data to grow exponentially. With too many information/options available at hand, people are facing the problem of choosing the right thing to buy, what to read, listen to, or watch next, how to spend their free time to entertain themselves, or even find the best person to date.

Recommender systems or recommendation systems (both are used interchangeably, but the term "recommender systems" will be used throughout the paper) comes into play at this point to support people to choose the right thing among the various options. Recommender systems provide methods or techniques to manage this overwhelming amount of data in a way to assist people to have more personalized choices based on their interests.

Various types of recommender system approaches are used in academic and commercial studies. Collaborative and content-based filtering algorithms are the most well-known and used recommender system approaches. Both collaborative and content-based filtering algorithms have strength and weaknesses. Hybrid methods, which incorporates multiple algorithms by trying to make use of the strength of each one, try to improve the performance of the overall recommender system.

The goal of this study is creating a hybrid recommender system by bringing an alternative approach by combining the negative similarity together with other recommender system techniques. In many of the recommender systems built so far, positive similarity has usually been used between users or items. Through using the negative similarity between users, an increase in the performance of the recommender system is aimed.

This thesis study focuses on building a recommender system for movie recommendation to users. In this thesis; MovieLens dataset, which was gathered over different periods of

time, is utilized to create the recommender system. The dataset includes ratings of movies given by the MovieLens users.

Apache Spark is an open-source library developed with Scala that allows us to process large scale datasets in parallel across clustered resources. It is capable of dealing with large-scale data processing and real-time analytics.

In the upcoming chapters of the thesis study, firstly, a literature review is provided about the subject. Then the information about the design of the model and the implementation of the design is given. Finally, the performance of the model is calculated and a conclusion is reached.

2. GENERAL INFORMATION AND LITERATURE REVIEW

2.1 RECOMMENDER SYSTEMS INTRODUCTION

Recommender Systems are commonly used especially in the e-commerce sector. Online businesses' arrival dramatically increased the entry of the product selection which an online business can offer. Over the course of the years, customers have had the ability to access immediately to thousands of movies on streaming video service, millions of items on e-commerce web sites, millions of news articles and content that are generated by users. The biggest e-commerce companies that have a very large customer base with a broad range of products place enormous value on making use of recommender systems for their businesses. They narrow down recommendable items/products that are relevant to the users' preferences or attention.

The classical brick and mortar stores have had a constraint of space which compels the amount of products they can hold. A business with a few numbers of products has a restricted amount of options to offer to its customers. Nowadays shelf capacity is no more a limitation and there is a great amount of unique products that the customer can dispose of. However, the consumption rate of the products is not the same and it varies.

There are well-known movies and there is a long tail of unknown or unheard movies that are demanded rarely. The long tail of movies tends to be watched by people with particular tastes. A person who is curious about thriller movies might have seen the Swedish-American psychological movie: *The Girl with the Dragon Tattoo*. This particular person might be highly curious about an unknown subgenre of Scandinavian Crime Fiction and might find the movies like *The Killing, The Bridge, Headhunters*, etc. The availability of this type of utilization is achievable when there is a large number of options for movies which can be used immediately. In some cases, the recommendation lists with the most popular items and typical editor's choices are simply not enough. If the enterprise can have more customer-product interaction, it can hold the customers engaged more to their business. Besides, licensing the most popular products inclined to be more costly than long-tail products. In such scenarios, holding the customer attached to the long tail product would be more rewarding.

Search engines provide useful results to find products in the long tail as long as the person knows what to search for. The information retrieval system comes in handy if the person searches for a distinct movie, such as *Headhunters.* In some circumstances, the person may look for "Scandinavian crime fiction" and retrieve a list of movies if the information retrieval system has that ability. Nothing is returned or recommended if the person is unfamiliar with such crime fiction subgenre or if the person does not know what to look for.

Many of the customers are best treated by their personal preferences, thus they may find the most related products for them among the vast amount of available items. If the business is able to forecast what will be their customer's next consumable, it will be for the benefit of both itself and its customer. Recommender systems take advantage of personalized experience.

Recommender systems have also been the center of attraction for academic researches. Sammut and Webb (2010) state that the most used and known algorithms within the machine learning algorithms are a recommender system. There are plenty of experiments conducted by researches to improve recommender system algorithms and to find the best platform. When the data become larger in size, different data processing tools are used in order to efficiently process this enormous amount of data.

Recommender Systems simplify people's life by helping them to find/choose what they are looking for among an almost endless list of options. People rely on other people's recommendations in almost every part of their life (Resnick and Varian 1997). For instance, people read other people's opinions and reviews for the products or items that they are interested in. Recommender systems have already been a complementary part of the biggest e-commerce businesses.

2.2 APPLICATIONS OF RECOMMENDER SYSTEMS

Recommender systems help people in many different areas to suggest to them what to buy, listen to, watch, read, who to follow, etc. They can serve people roughly in three different areas.

2.2.1 Product Recommendations

E-commerce sites or any business which has a vast number of products might take advantage of recommender systems. Product recommendations are one of the first recommender system applications in the digital world (Schafer, Konstan, and Riedl 2001, pp.115-153). E-commerce sites try to create a list of recommendation by predicting the most appropriate products for their customers and to provide that listing in as many places as possible such as the homepage, specific item page, shopping cart, checkout page, etc. The recommendations for the customer are based on the former clickstream data or past purchases of the related customer. E-commerce sites usually augment the recommendation list with the most popular items, recently added products, and promoted products. When the customer views a product and if he/she has the intention to buy it, the system shows a list of relevant items or "*customers who bought this also bought that*" type of items that the customer might have interest based on his/her profile. Similar features may be used by offline businesses, to some degree. Custom-made marketing proposals, customer-related discount tickets, and dynamically changing call-center scripts are among the methods the recommender systems are utilized for offline experiences.

2.2.2 Content Recommendations

A customized news feed is one of the methods to increment the cohesion of the content creating sites. Users usually have restricted time to read or watch the content and therefore the content creating sites are highly motivated to engage the users on their site by providing them the most related news, or a friend's post on social media or a tweet. Almost all of these suggestions have the grounds on the current user and other users' previous behaviors on the same site (Thorson 2008, pp.473-489).

2.2.3 Entertainment Recommendations

Binge-watching is described as watching a single TV show or a streaming media show for a long period. It has become trendy with the outbreak of video streaming services such as Hulu, YouTube TV, and Amazon Prime Video. The consumption of digital entertainment items such as videos, or music has happened to be the ruling method the way they are spent. On-demand music businesses such as Spotify, Deezer, and Amazon Prime Music have a tremendous collection of songs and supply good suggestions to the user according to the user's tastes. They are capable of suggesting a new artist's song to a user because it has similar features with the songs the user has already liked.

Netflix announced a public contest called the Netflix Prize to award the best recommender system algorithm that could surpass its own recommendation system – CineMatch. Although the recommender systems had existed before, this led the studies on the recommender system to focus on user-item interactions (Koren, Bell, and Volinsky 2009, pp.30-37).

2.3 RECOMMENDER SYSTEM APPROACHES

There are various approaches available for building recommender systems. For example, (i) in popularity-based recommender approach, recommendations are made by using items that are high in-demand, (ii) in Collaborative Filtering (CF), templates of user preferences are explored in order to provide customized recommendations (Breese, Heckerman, & Kadie, 1998); (iii) in Content-based technique (CBF), the suggestions are built by using the akin features the user has previously liked. All of the recommender system approaches have strengths and weaknesses (Burke 2002). For that reason, some recommender systems use more than one approach in a hybrid way in order to improve the results by combining the strengths of different approaches (Adomavicius and Tuzhilin 2005).

2.3.1 Collaborative Filtering

The principle concept of collaborative filtering technique is to take advantage of the data regarding the prior act or the viewpoint of existent users in a group to forecast the products might concern the user most or liked by the user (Jannah et al. 2011, p.43). The concept that when multiple users rated the items in a similar way in the past, they would probably act in a similar way in the future, form the basis of collaborative filtering (Breese, Heckerman, and Kadie, 1998). If, for instance, one of the two similar users likes a movie he/she watches, then it is a positive sign that the other user will have a similar taste for that movie. This technique is quite effective since it does not need further information about the items like genre, director, cast, etc. to provide recommendations.

In collaborative filtering technique, a user-item matrix is created from users' ratings for items. Later, users alike are paired by measuring the similarities of their user profiles in order to build recommendations. These similar users create a group that is called neighborhood (Herlocker et al. 2004, pp.5-53). An item that received positive ratings from users in the close vicinity and has not previously been experienced, is recommended to the user.

Collaborative filtering technique might be classified into two types: Memory-based and model-based (Bobadilla, Ortega, and Hernando, 2013).

2.3.1.1 Memory-based approach

In this approach, the data collected from the user ratings are utilized to calculate the resemblance to the items or users. The items, previously rated by the user, have an important role in determining a neighbor that has similar preferences in common (Zhao and Shang 2010). When a user's neighbor is discovered, various methods might be utilized to incorporate neighbors' choices to build recommendations. Since these techniques are effective, they have gained extensive accomplishments in real-life applications. These techniques might be categorized into two major groups: user-based techniques and item-based techniques (Liang et al. 2016, pp.975-1004). User-based technique strives to discover the closest users to the specified user, $u_{i,}$ by using the ratings'

resemblance as a basis and suggest the items that are enjoyed by the closest users to the specified user, u_i. The principal concept of this technique is to determine the users that have similar ratings and recommend the user u_i the products that haven't been viewed or evaluated by him/her yet and positively evaluated by similar users. (Hahsler, 2014). Conversely, Item-based technique uses the resemblance of items rather than the resemblance of users for calculating the predictions. The specified user's evaluated products are extracted from the matrix, and a pattern of resemblances is built. Then the technique tries to identify the resemblance of the specified product to the retrieved products. Finally, it chooses *k* most similar products. The weighted average of user u_i's rating on similar items forms the basis of the prediction. The item-based technique is interested in the features of the product p_j, and tries to find the products with the most similar features to product p_j. Finally, the suggestions are made by utilizing similar products (Liang et al. 2016, pp.975-1004).

Various types of similarity metrics might be utilized to compute the resemblance of users/items. Correlation-based and Cosine-based similarity measures are among the most popular similarity measurements (Ricci, Rokach, & Shapira, 2011). In order to compute how strong the relationship between two variables, the Pearson correlation coefficient might be utilized. (Melville, Mooney, & Nagarajan, 2002).

$$S(i,k) = \frac{\sum_j (v_{ij} - \overline{v_i})(v_{kj} - \overline{v_k})}{\sqrt{\sum_j (v_{ij} - \overline{v_i})^2 \sum_j (v_{kj} - \overline{v_k})^2}} \tag{2.1}$$

In Equation 2.1, *S(i,k)* measures the resemblance of users u_i and u_k. v_{ij} represents the evaluation which user u_i provided to product p_j. User u_i's average evaluation is represented by $\overline{v}_i$.

Given in Equation 2.2, the cosine similarity of two variables is a metric utilized to compute the cosine of the angle between them.

$$cos(u_i, u_k) = \frac{\sum_{j=1}^{m} v_{ij} v_{kj}}{\sqrt{\sum_{j=1}^{m} v^2{}_{ij} \sum_{j=1}^{m} v^2{}_{kj}}} \quad (2.2)$$

By using this similarity score, it is possible to check every single user and the remaining users. If the similarity between vectors is high, then the similarity between users is high, too. Therefore, we get a symmetric matrix with all of the users' similarity scores as defined by the matrix in Figure 2.1.

Figure 2.1 Similarity score matrix

$$S = \begin{array}{c} \begin{matrix} u_1 & u_2 & & u_i & & u_n \end{matrix} \\ \begin{bmatrix} 1 & S(1,2) & \dots & S(1,i) & \dots & S(1,n) \\ & & & & & S(2,n) \\ & & \ddots & & & \\ & & & 1 & & \vdots \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix} \end{array} \begin{matrix} u_1 \\ u_2 \\ \\ u_i \\ \\ u_n \end{matrix}$$

Initially, the users that are greatly relevant to the specified user, u_i, are required to be determined. This can be achieved by choosing *k* users greatly resembling the specified user, u_i. Then the items liked by similar users are determined, and the movies the user has already watched are eliminated. Then, the ones which have been seen by the greatly related users are measured by utilizing the resemblances. The forecast that shows the ratings which the specified user, u_i, might provide to the movies is reached as the ultimate outcome. Subsequently, *N* movies that have the greatest ratings in the forecasted list is selected (Isinkaye, Folajimi, and Ojokoh, 2015). The formula is shown in Equation 2.3.

$$p(i,k) = \bar{v}_i + \frac{\sum_{i=1}^{n} (v_{ij} - \bar{v}_k) \, x \, S(i,k)}{\sum_{i=1}^{n} S(i,k)} \quad (2.3)$$

2.3.1.2 Model-based approach

In this approach, past evaluations are utilized to learn a pattern and they are used to increase efficiency. The operation of discovering the pattern might be made by utilizing data mining or ML algorithms. Singular Value Decomposition (SVD), Clustering, and Regression are examples of these approaches. These approaches study the user-item matrix in order to detect the connection among items. These connections are utilized to check the top-N recommendation list. These techniques find a way out for the sparsity problems of recommender systems. With the usage of learning algorithms, the way the recommendations are done: not only what to consume but also when to consume is aimed to be recommended.

2.3.1.3 Advantages and disadvantages of CF

CF technique has several important benefits. In areas where it is difficult to analyze the content by computing systems and where there are not many contents related to the items, the collaborative filtering technique can provide successful recommendations. It is capable of producing serendipitous suggestions, which indicates that it has the ability to suggest products which are related to the user despite the fact that the content itself is not in user's profile (Schafer et al. 2007, pp.291-324). Although CF technique is widely used and has considerable success, it has some potential disadvantages.

2.3.1.3.1 Scalability

Scalability is a problem related to recommendation algorithms since the calculation becomes linearly bigger the number of items and users increases (Park et al. 2012, pp.10059-10072). While a recommender system may provide sufficient recommendations when the dataset is relatively smaller, it might provide poor recommendations if the dataset becomes bigger. Therefore, it is significant to implement recommendation methods which are able to scale up well while the dataset is becoming very large. Methods for reducing the dimension including Singular Value Decomposition

(SVD) technique are utilized for overcoming the aforementioned scalability problem in order to provide trustworthy and effective recommendations.

2.3.1.3.2 Cold-Start

The cold-start problem indicates a condition in which the recommender system lacks enough data about an item or a user to create proper predictions (Burke, 2007). This is one of the main drawbacks that decrease the success of recommendation. It is not possible to properly create the item or user profile because the item/user is not known by the system and it is not possible to know the preferences of the new user.

2.3.1.3.3 Synonymy

Synonymy refers to the inclination of items closely alike having various records or names. Generally, it is hard to comprehend for the recommender system to differentiate strictly related items like the distinction between e.g. men's wear and men's clothing. In such cases, collaborative filtering algorithms normally find no relationship between these terms in order to calculate their similarity. To solve the synonymy problem, various methods such as SVD, creating a thesaurus, and automatic term expansion have the capability to solve this difficulty. The main drawback of these methods is that they might add unintentional different meanings that often result in the deterioration of the recommendation performance.

2.3.1.3.4 Data Sparsity

Data sparsity happens because of the absence of adequate data, namely the majority of the available products in the database have not been evaluated by users (Park et al. 2012, pp.10059-10072). That always results in a scattered user-item matrix. This leads to the failure to determine the neighborhood and finally, the creation of poor recommendations. Furthermore, data sparsity results in problems of coverage, which can be described as the proportion of items within the structure for which the recommendations can be provided (Su and Khoshgoftaar 2009).

2.3.2 Content-based Filtering

CBF technique utilizes the explicit (rating) or implicit (clicking on a link, etc.) data which the user produces. By using the user-produced data, a user profile is generated and it is utilized to create suggestions to the user. The more inputs are provided, the better and more accurate recommendations the user gets.

In CBF technique, user profiles form the basis of suggestions by using the items' extracted properties for the content that the user has previously evaluated (Burke 2002). The items highly relevant to ones evaluated positively by the user are considered to be the candidate recommendations to the user. In order to generate significant recommendations, various kinds of models are utilized in CBF (Isinkaye, Folajimi, and Ojokoh, 2015). Those algorithms provide suggestions by finding out the principal model with machine learning algorithms or statistical analysis.

The profile of another user does not have any impact on the recommendation since the recommendation is made by using the content of the item. Therefore, another user's profile is not required in the CBF technique. Likewise, should the user profile somehow change, it is still possible in CBF technique to adjust the recommendations.

2.3.2.1 Advantages and disadvantages of CBF

CBF technique circumvents some of the defects of CF techniques. This technique is able to suggest fresh items though the users have not provided any ratings. Therefore, even though there aren't any user preferences in the database, the success of the recommendation cannot be influenced. Furthermore, when the user's interests change, it is still capable of adapting its recommendation rapidly. Content-based filtering technique is able to deal with circumstances in which users do not have a neighborhood with other users and only have a user profile extracted from the basic features of the items. Users do not require to share their profile to get recommendations, which protects their privacy (Shyong, Frankowski, and Riedl 2006, pp.14-29). Content-based filtering technique can yield explanations to users about how recommendations are created. Nevertheless, content-based filtering technique faces different drawbacks as revealed in the literature

(Adomavicius and Tuzhilin 2005). CBF technique depends heavily on the items' metadata. That requires a very well-formed user profile and sufficient item descriptions to supply suggestions to the user. That is described as the analysis of the restricted content. For this reason, the efficiency of the CBF technique heavily counts on the descriptive data's availability. Over-specialization of the content is another important drawback of content-based filtering technique (Zhang and Iyengar 2002, pp.313-334). That means the recommendations provided to users are confined to the items alike.

2.3.3 Hybrid Filtering

Hybrid filtering method incorporates multiple recommendation methods to create improved recommendations by evading some of the restrictions and problems of plain recommendation systems (Adomavicius and Zhang 2012) and (Stern, Herbrich, and Graepel 2009, pp.111-120). The main theory behind hybrid filtering technique is having a mixture of recommender methods would produce more precise and efficient suggestions than an individual recommender method because, with the hybridization, defects of one method might be reduced through utilizing the benefits of the other. (Schafer et al. 2007). Utilizing various recommender methods might overcome the defects of a single method when used combined. Hybridization might be achieved by one of the subsequent approaches: applying the methods one by one and incorporating the outcome, applying CF methods in CBF methods, applying CBF methods in CF methods, and building a consolidated model which combines methods together.

2.3.3.1 Weighted Method

In the weighted method, various recommendation algorithms' outcomes are brought together to create a recommendation or prediction list by combining the results of each algorithm by using a linear formula. An item's rating is calculated as the weighted total of evaluations extracted from a group of recommenders. The weights are concluded by training on earlier user ratings and they may be adapted as new ratings appear. P-tango can be considered to be a sample of this type of hybridization (Claypool et al. 1999). P-tango includes a CF and CBF recommender. In the beginning, they are granted weights

equally, but they are modified later based on the newer ratings. The advantage of the weighted hybridization is that each recommender technique's benefits are applied simply at the event of recommendation operation.

2.3.3.2 Switching Method

In switching hybridization method, various measures are utilized for swapping the recommendation methods. This method has the capability to refrain drawbacks particular to one algorithm by switching to another algorithm. The advantage of this method is that it is responsive to the weaknesses and the strengths of its composing recommender algorithms. The principle drawback of switching hybridization method is that it generally brings more complexity to the course of recommendation. It is because the switching criteria grow the amount of parameters, and they must be decided (Burke 2002). The DailyLearner (Billsus and Pazzani 1999, pp.99-108) system, which is an example switching hybridization, applies content-based and collaborative switching. It initially uses CBF recommendation and unless it can build a good recommender system, it switches to the CF algorithm.

2.3.3.3 Cascade Method

The cascade method utilizes a repetitive improvement procedure to build a sequence of preference between different items. One method's recommendations are refined by other method's recommendation. The first recommendation method results in a rough list of recommendations. Then it is improved by the following recommendation method. The cascade method is quite effective as well as permissive to noise because of the coarse-to-fine structure of the repetition. An instance of the cascade hybridization is EntreeC method (Burke 2002) which uses a knowledge-based algorithm and collaborative algorithm in a cascade way.

2.3.3.4 Mixed Method

In this method, the outcomes of various recommender techniques are combined simultaneously instead of getting just a single suggestion for each item. Each item gets various suggestions related to itself from different recommender methods. In this method, individual efficiency of the recommender system does not always influence the overall performance. PTV is a sample of mixed hybridization method (Smyth and Cotter 2000, pp.53-59) that recommends a schedule to a user for watching TV by mixing recommendations from collaborative and content-based systems. PickAFlick (Burke, Hammond, and Young 1997, pp.32-40) and Profinder (Wasfi 1999, pp.57-64) are other examples of this type of recommendation method.

2.3.3.5 Feature Combination

In this method, the rating generated by the first algorithm is used as a feature in other recommendation algorithms. The usual implementation of this method is to provide the rating of a collaborative filtering algorithm to a CBF algorithm because the CBF algorithms heavily rely on the features of the items. Pipper (Basu, Hirsh, and Cohen 1998, pp.714-720) can be considered as a sample of this method. It utilizes the ratings of CF technique as an additional feature in the CBF technique in order to recommend movies. The advantage of this method is that most of the time it does not depend entirely on the data derived from the collaborative filtering technique.

2.3.3.6 Feature Augmentation

The feature augmentation method takes advantage of the evaluations and some other additional knowledge created by a former algorithm. Additionally, it needs further capabilities of the recommender algorithms. The Libra System (Mooney and Roy 2000, pp.195-204), creates CBF book recommendation on the basis of the information available at Amazon web site through making use of a naïve Bayes text classifier. Feature augmentation method is better than the feature combination methods because it augments a limited number of features to the main recommender algorithm.

2.3.3.7 Meta-level Hybridization

In this method, the recommendation algorithms can be combined by utilizing the model created by one as the contribution for another. The model created by one of the recommendation algorithms has normally more information than an individual rating. Meta-level (Pazzani 1999, pp.393-408) hybridization method is capable of resolving the collaborative filtering technique's drawback of sparsity through utilizing the complete model which the first algorithm discovered as the input to the next algorithm. The Fab (Balabanovic and Shoham1997, pp.66-72), which is a web filtering system, is an example of meta-level hybridization method.

2.4 ALGORITHMS USED

2.4.1 Pearson Correlation Coefficient (PCC)

Pearson Correlation is a statistical method used to determine whether there is a linear relationship between two numerical measurements. If there is any relationship, it is also used to determine the direction and severity of this relationship. Pearson Correlation Coefficient is denoted by *r*. Essentially, it tries to draw the most appropriate line between two variables. PCC shows how distant those points settle around that line.

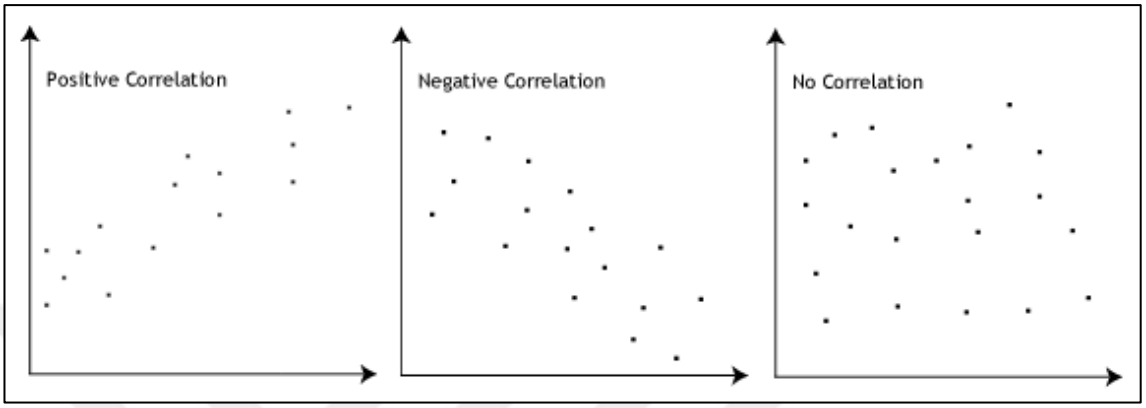
PCC might take a value between -1 and +1. If the value is 0, it indicates that the two variables are not correlated at all and often called zero-order correlation. If the value is bigger than 0, it indicates a positive correlation and the value +1 represents a highly positive correlation. If the value is smaller than 0, it indicates a negative correlation and the value -1 symbolizes highly negative correlation.

The kind of correlation might be categorized by analyzing how one variable behaves or changes while the second variable increases:

i. If it rises as well, it signifies a positive correlation,
ii. If it reduces, it signifies negative correlation,
iii. If it does not change, it signifies zero correlation.

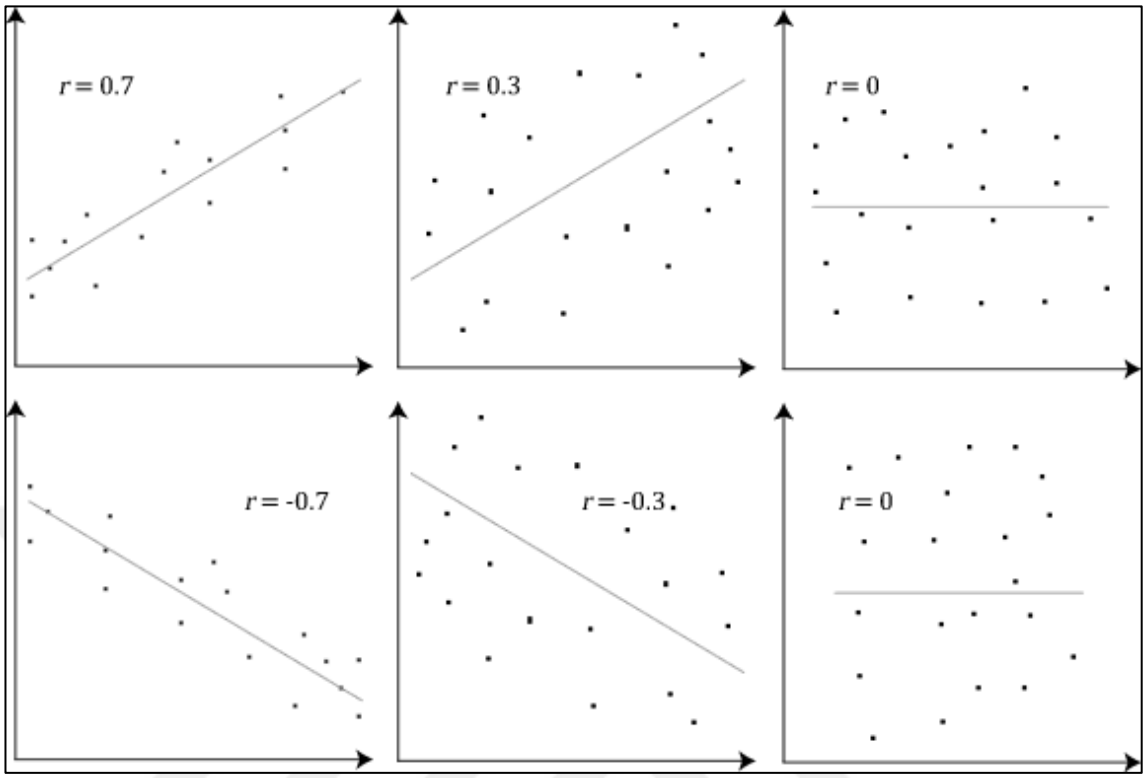
In Figure 2.2, different types of correlations, positive, zero, and negative correlations, are depicted:

Figure 2.2: Correlation examples



If the correlation between the variables is stronger, PCC would be near to +1 (if it is a positive correlation) or -1 (if it is a negative correlation) based on the correlation type, positive or negative, respectively. If the Pearson correlation coefficient is +1 or -1, that would mean that each data point fits the best fit line. This indicates that no deviation from the best fit line is in place. The Pearson correlation coefficient values between +1 and -1 (r=0,7 or r=0,3 for example) show some deviation occurs in the vicinity of that most appropriate line. If PCC is near to the value of 0 that would mean the variation is greater in the vicinity of the best fit line. In Figure 2.3, different correlations and their correlation coefficient values are shown.

Figure 2.3: Various correlations and correlation coefficients



2.4.2 k-Nearest Neighbors (kNN)

kNN algorithm has been in use to get recommendations from the beginning of recommendation techniques onward (Albadvi and Shahbazi, 2009, pp.11480-11488). It is still commonly used since it is easy to understand and comprehend. In the kNN algorithm, the main idea is to get the predictive value of a pair of one user-item by utilizing the weighted average ratings of the pair's neighbors. It includes three steps:

i. Formation of the similarity: In this step, the computation of similarity between items or users are made. Cosine similarity and Pearson correlation are widely used as similarity measures.
ii. Selection of the neighbor: in this step, after the calculation of the similarity matrix, the neighbors are selected for the target item or user. In the user-based kNN model, the target user's neighborhood is built through discovering the target user's most similar users. Likewise, in item-based kNN model, the neighborhood can be built through discovering the target item's most similar items. This method is

generally called best-n-neighbors (Cho and Kim 2004, pp.233-246). Most of the time, the same neighbor selection method is in common among kNN models.

iii. Generation of the prediction: In the last step the predictive ratings of target user to all items are built. Also, the ratings that have the highest values are turned back to the target user as the recommended item set.

2.4.3 Matrix Factorization

There are two identified evident constraints in kNN algorithm. These are the “Cold start problem”, and the “popularity bias”. If the training dataset exceeds to fit in one node, there might be another constraint called “scalability problem”. The problem of cold start refers to a situation in which a new user or item introduced to the system and the system lacks the necessary data to create recommendations. The system is not capable of making a decision about any deductions from the user or item if it does not have adequate data regarding the user or item. It is one of the big deals for the recommender systems since new users or items are constantly included in the system. Popularity bias means that the system builds recommendations for the movies which are rated mostly by the users, without any personalization. Scalability problem means the difficulty when scaling to bigger datasets while new items and users are constantly joined to the system.

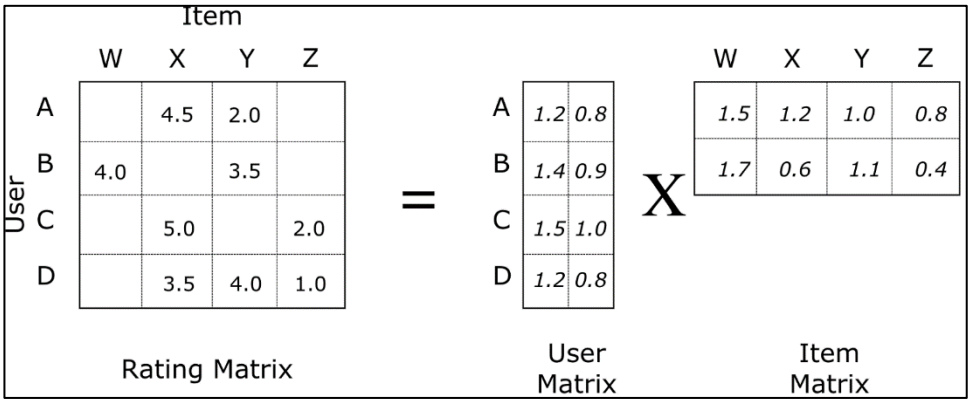
The three main problems mentioned above are characteristic drawbacks of collaborative filtering technique. They naturally come along with the user-item interaction matrix where each value represents a value of rating of users ***i*** to item ***j***. In almost all of the MovieLens datasets, most of the movies have hardly any ratings or even not rated at all by users. This results in having an intensely sparse matrix with most of the movies have missing ratings as shown in Figure 2.4.

Figure 2.4: Sparse matrix

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

In Collaborative filtering technique, matrix factorization can be used to solve data sparsity problem in such sparse matrices. Matrix factorization has become widely known after its usage in the Netflix Prize Challenge.

Figure 2.5: Matrix factorization



Matrix factorization, as shown in Figure 2.5, is just a group of a mathematical process for matrices in linear algebra. To be more precise, it is a matrix's factorization into a product of matrices. In the context of the collaborative filtering algorithm, matrix factorization algorithms operate by breaking down the user-item matrix into a product of two

rectangular matrices that have lower dimensionality. One of the matrices is the item matrix. In this matrix, the columns indicate items while rows indicate latent factors. The second matrix is the user matrix. In this matrix, columns indicate latent factors and rows indicate users. Matrix factorization solves the data sparsity problem as follows:

i. In the model, the rating matrix is factorized into movie and user representations. This lets the model providing more accurate and ratings, which is more personalized, to the users,
ii. With the help of matrix factorization, the little-known movies may have abundant latent representations inasmuch as the popular movies have. This develops the recommender system's capability to recommend less-known movies.

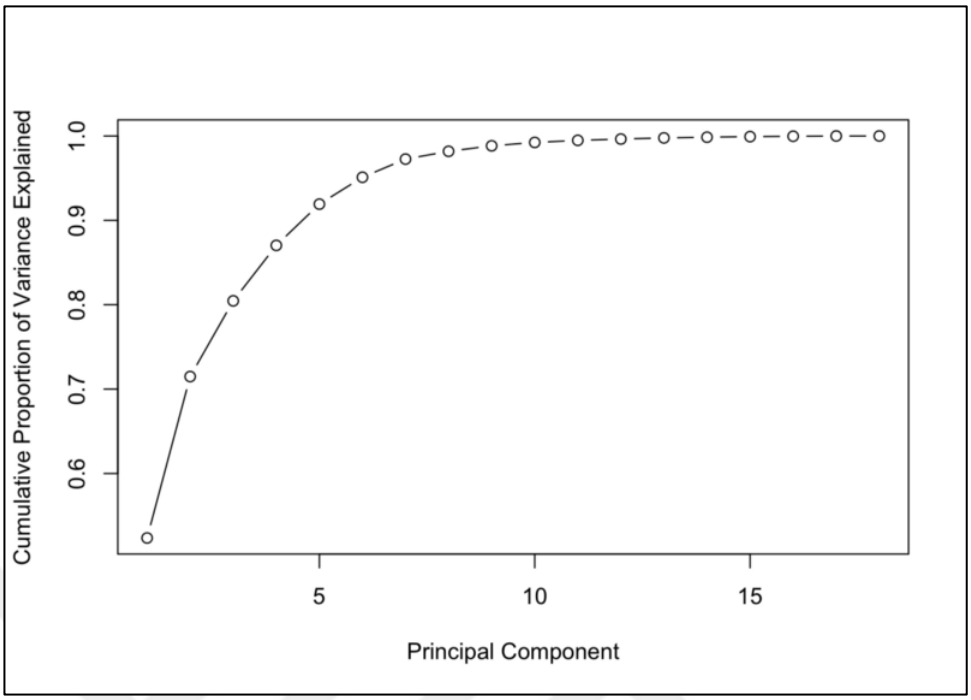
In the sparse user-item matrix, forecasted rating (*f*) of user *u,* for the item *i* is calculated as shown in Equation 2.4:

$$\bar{r}_{ui} = \sum_{f=0}^{n\ factors} H_{u,f} W_{f,i} \tag{2.4}$$

The rating of user *u* for the item *i* might be described as the dot product of the item and user latent factor respectively.

In the aforementioned formula, the number of latent factors may be tuned via cross-validation. Latent factors are the properties in the lower dimension latent space estimated from the user-item matrix. The main opinion that forms the basis of matrix factorization is utilizing latent factors to display movie topics or user preferences in a much lower dimension space. It is one of the most efficient dimension reduction methods in machine learning.

Figure 2.6: Variance Explained By Components in PCA



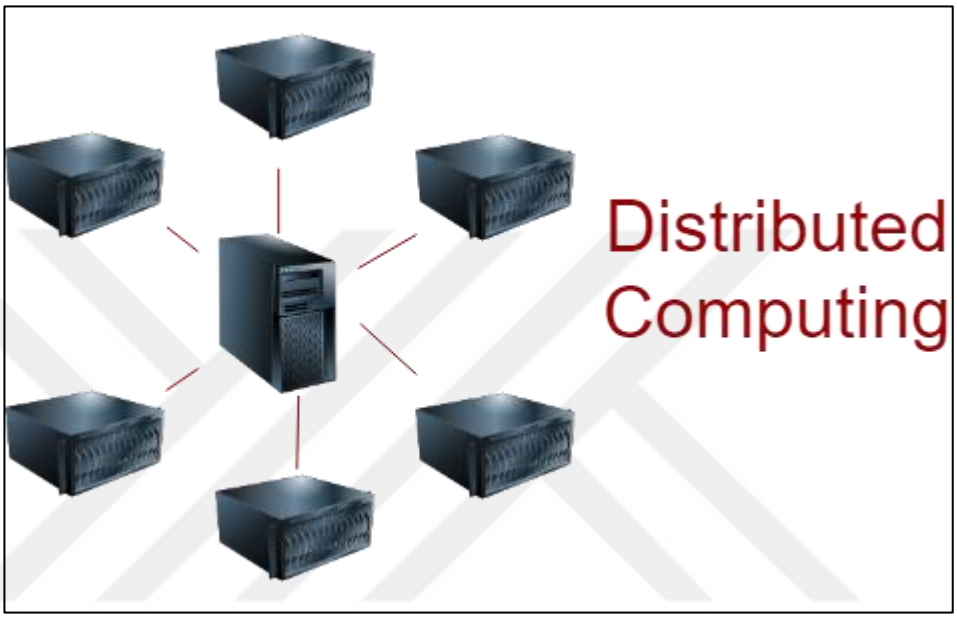
Similar to the idea of components in Principal Component Analysis (PCA), the amount of latent factors identifies the volume of abstract information that is wanted to be held in a lower dimension space. A matrix factorization with a single latent factor equals to the most popular recommendation. In order to make the personalization better, the number of latent factors can be incremented to the extent where the model begins to overfit. One of the widely used methods to prevent overfitting is to increase the regularization terms for the target function.

The main goal of matrix factorization is decreasing the error between true rating and predicted rating to the minimum degree. When there is an objective function at hand, a training routine (e.g. gradient descent) is required to finalize the practice of a matrix factorization algorithm. This practice is in fact called Funk SVD because Simon Funk who has shared his outcomes with the research community in the course of Netflix prize challenge.

While Funk SVD was quite efficient in matrix factorization with a single computer at that time, it is not scalable since the volume of the data increases today. With terabytes or

even petabytes of data, it is almost impractical to put the data with such size into one computer. Therefore, a machine learning model (or framework) is required to train that big amount of dataset that is distributed across a cluster of computers as shown in Figure 2.7.

Figure 2.7: Scaling ML Applications with Distributed Computing



2.4.3.1 Alternating Least Squares (ALS) Algorithm

ALS is kind of an algorithm that is used to factorize a matrix. ALS algorithm can be parallelized, which is one of the main benefits of this algorithm. This algorithm can be applied in Spark and it is developed for CF problems that are largely scaled. ALS performs relatively well when it comes to solving scalability and sparsity problems of the dataset. It is easy and understandable and it can scale well to enormous datasets.

Similar to other machine learning algorithms, ALS has a group of parameters peculiar to itself. These parameters are tuned by cross-validation and hold-out validation.

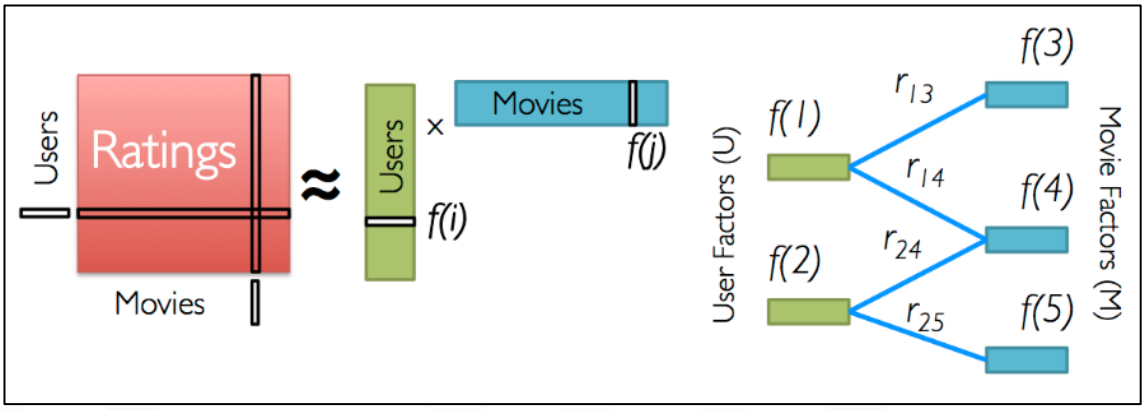
Alternating Least Square (ALS)'s main parameters are:

i. *numBlocks* specifies the amount of chunks utilized to parallelize computation,
ii. *maxIter* specifies the amount of repetitions the program will run at a maximum,
iii. *regParam* is ALS's regularization parameter,
iv. *rank* specifies the amount of latent factors in the model,
v. *nonnegative* specifies the usage of nonnegative restraints for least squares

Alternating Least Squares algorithm is an optimization process that runs iteratively. In each iteration process, the aim is to come closer to a factorized representation of the original data.

In the ALS implementation for the MovieLens dataset, the algorithm initially loads the users matrix randomly with ratings and then optimizes the rating of the movies in a way that the error is minimized. Then, the algorithm keeps the movies matrix constant and optimizes the rating of the users matrix. The name "Alternating" comes from this alternation of the optimization process between matrices. Given a constant set of user factors, the given ratings are used to discover the best-predicted ratings for the movie factors. Then the alternation happens and the best user factors for a given set of movies are picked. The outline of this algorithm is shown in Figure 2.8.

Figure 2.8: Alternating Least Squares algorithm



2.5 EVALUATION OF RECOMMENDER SYSTEM TECHNIQUES

A recommender system's efficiency might be calculated by utilizing various kinds of measures that might be the coverage or accuracy. The kind of measurements applied relies on the type of recommender algorithm used. According to ISO 5725-1, accuracy represents the proximity of a measurement to the true value. For the recommender systems, the accuracy describes the ratio of total possible recommendations to the correctly made recommendations. Coverage, on the other hand, calculates the portion of items in the search space in which the method is capable of producing recommendations. The measurements to calculate the algorithm's accuracy are bisected as decision support accuracy and statistical measurements (Sarwar et al. 2001). The availability of each measurement relies on the features of the data used to create recommendations (Friedman, Geiger, and Goldszmidt 1997, pp.131-163). Statistical accuracy measurements assess the recommendation algorithm's accuracy by checking the forecasted rating and real rating of the user. Root Mean Square Error (RMSE), Mean Absolute Error (MAE) (Goldberg et al. 2001, pp.133-151), and Correlation are generally utilized as measurements of statistical accuracy. MAE measures the variance of suggestion from the actual rating of the user. MAE is calculated as shown in Equation 2.5 (Claypool et al. 1999):

$$\mathrm{MAE} = \frac{\sum_{p=1}^{n} |y_p - x_p|}{\mathrm{n}} \qquad (2.5)$$

In the above formula, y_p refers to the forecasted assessment for user u on product p, x_p refers to the real assessment and n refers to the aggregated amount of assessments for the product in a dataset. If the MAE value is smaller, it indicates that the recommendation algorithm provides better forecasts for user assessments.

RMSE, which is given by (Cotter and Smyth 2000, pp.957-964), is calculated as shown in Equation 2.6:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{2.6}$$

RMSE gives additional importance on bigger absolute error. If the RMSE value is smaller, it indicates that the recommendation algorithm is more accurate.

Coverage represents the items' and users' proportion which the recommendation might produce forecasts. The forecast might be almost impractical to generate if no user of a limited number of users provided ratings to an item. If items' or users' neighborhood scope is delimited to a smaller one, coverage might be decreased (Papagelis and Plexousakis 2005, pp.781-789).

2.6 APACHE SPARK

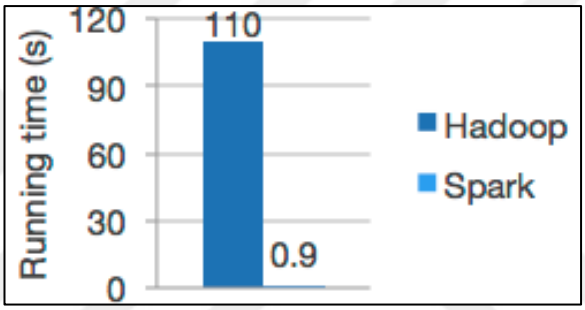
Apache Spark is a widely used unified analytics engine to compute the scattered data and to process the data that is largely scaled. It is a computation engine for analyzing largely scaled datasets which are generally unable to fit into the memory of a single computer or node.

In 2009, Spark began as a research project at AMPLab in the University of California, Berkley. In 2010, it became an open-source project, and in 2013 it became a part of Apache Foundation. Apache Spark is written in Scala and for that reason, it needs a JVM

to run. It is possible to use Python, R, and various other languages in Spark through its high-level API.

Of the many principal reasons, Apache Spark is so widely used is its speed. According to its website (spark.apache.org), it is 100 times faster than Hadoop. An example of comparison is shown in Figure 2.8. It obtains this high performance for both streaming and batch data by using a physical execution engine, a query optimizer, and a state-of-the-art Directed Acyclic Graph (DAG) scheduler.

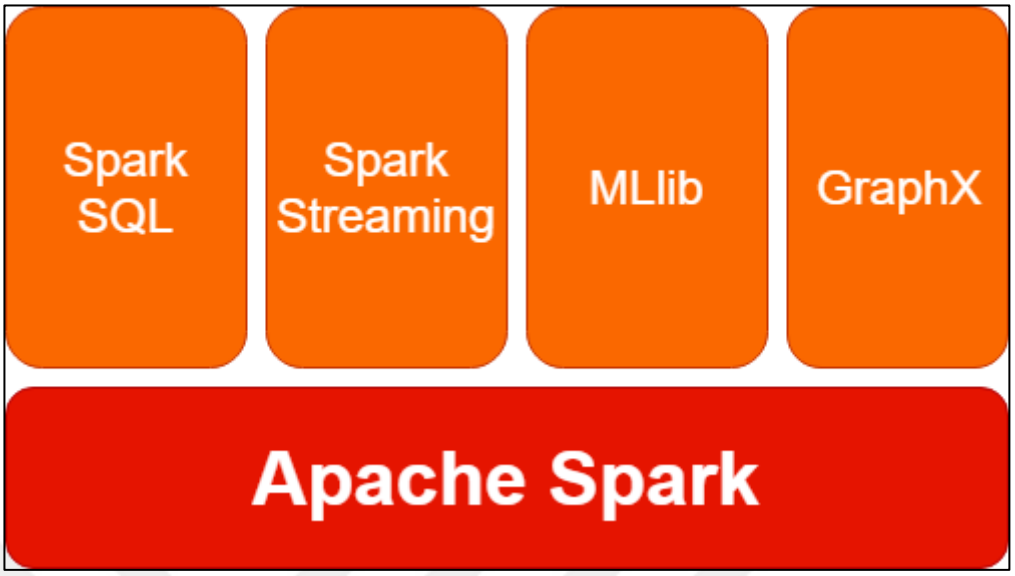
Figure 2.9: Comparison of running time between Spark and Hadoop



2.6.1 Spark-Core

Spark core processes distribution work across worker nodes and other fundamental works such as fault recovery. Higher-level libraries such as MLlib for Machine Learning, GraphX, Spark SQL, and Spark Streaming run on top of Spark core engine as shown in Figure 2.9. These libraries might be consolidated smoothly in one application.

Figure 2.10: Apache Spark Architecture



2.6.2 MLlib

MLlib is a library for ML that can be scaled. Its main objective is to make machine learning simple and scalable. It includes a bunch of built-in algorithms like clustering, regression, classification, and recommendation (alternating least squares - ALS).

2.6.3 Spark SQL

Spark SQL is a module for managing structured data. With Spark SQL, it is feasible to query structured data by utilizing either structured query language or a similar API. It can be used with Python, R, and similar languages.

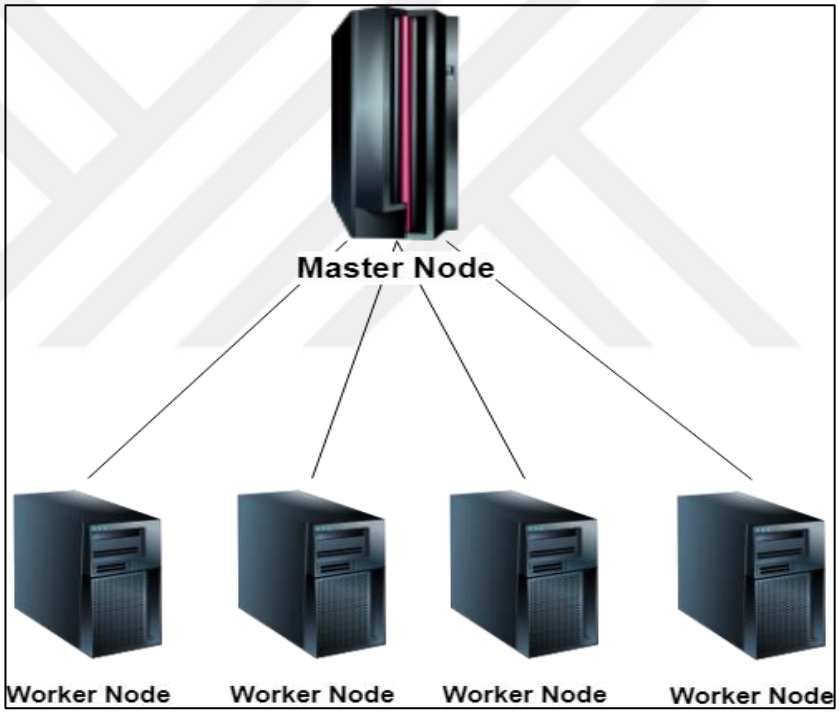
It ensures uniform data access. SQL and DataFrames supply a common way to connect to various data sources, including JDBC, Hive, JSON, Parquet, etc.

Spark SQL can scale up to hundreds of nodes simultaneously by utilizing the Spark framework.

2.6.4 Distributing Computing in Apache Spark

In Apache Spark, scaling can be either horizontally or vertically. In horizontal scaling, weak nodes are joined to a cluster. Similar to Figure 2.10, in a typical cluster, there are various worker nodes and a master node. Master node manages how jobs are divided across the worker nodes. Horizontal scaling is fault-tolerant and it can be expandable. In vertical scaling, more components such as RAM are attached to one worker node. This will ultimately hit a limit; therefore, it is limited to expansion. Vertical scaling is not fault-tolerant.

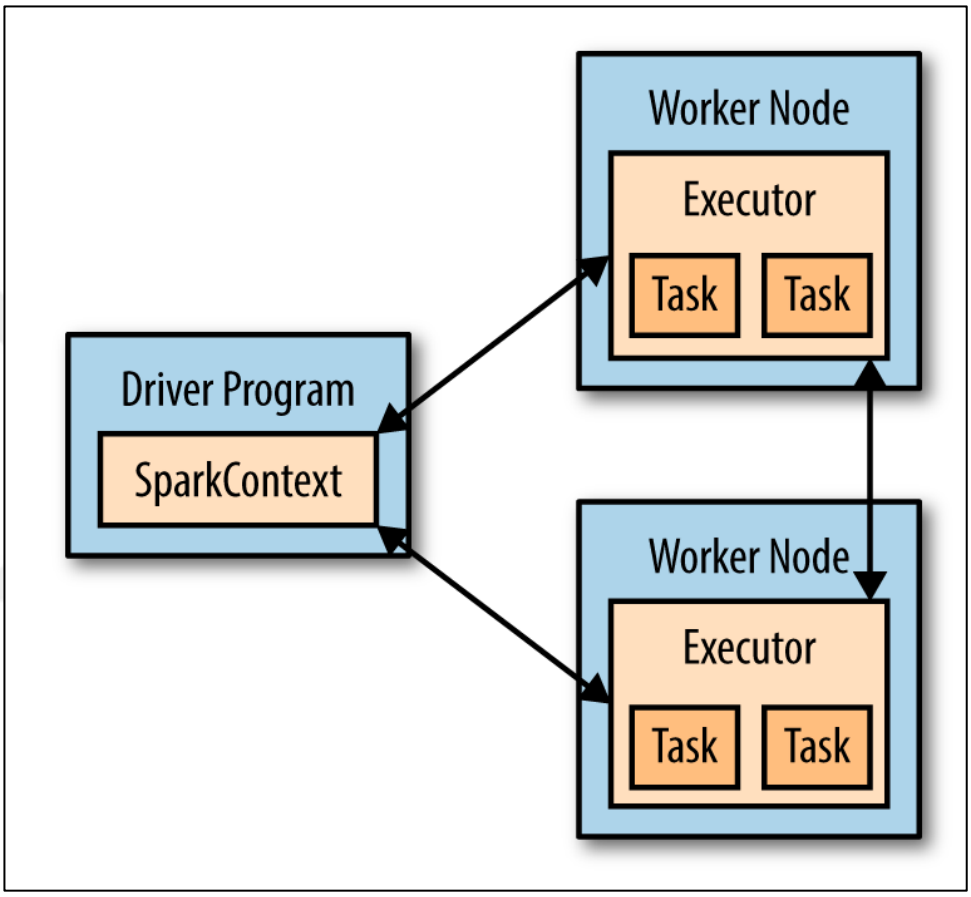
Figure 2.11: Master and worker nodes



In Apache Spark, each Spark application includes a driver program that initiates various parallel processes in the cluster. This program resides on the master node and manages the parallelism by accessing Spark through a SparkContext object, which is a means of connecting to a computer cluster. SparkContext is the access point to Apache Spark's services. The SparkContext is basically a client of the execution environment of Spark and it performs as the master of the Spark application.

In order to run the operations for distributing computing, deriver programs usually operate a number of worker nodes called executors as shown in Figure 2.11. The tasks that are operated in parallel are divided across these worker nodes. The tasks are automatically distributed in the Resilient Distributed Datasets (RDDs) by Apache Spark.

Figure 2.12: SparkContext and Executors



2.6.5 Resilient Distributed Datasets (RDDs) and DataFrames

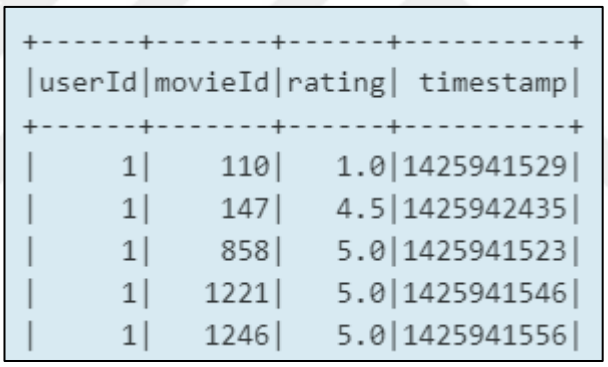
In Apache Spark, there are various abstractions for data: Resilient Distributed Datasets (RDDs), DataFrames, Datasets, and SQL Tables. All of these various abstractions show distributed collections of data.

RDD was the main API in Spark since its beginning. RDD is an unchangeable distributed compilation of data. RDD is split over nodes in the cluster and might be used simultaneously.

From the Apache Spark version 2.0 onwards, DataFrames have been the principal API in Spark. DataFrame's syntax is more instinctive than of RDD's, but their functionality doesn't differ. RDDs are part of the low-level API and the DataFrames are part of the Structured APIs (Zaharia and Chambers 2018). Similar to an RDD, a DataFrame is an unchangeable distributed compilation of data. Different from an RDD, in a DataFrame, data is formed into named columns. The RDD functionally and visually looks like to the Pandas in Python and R DataFrames. It is also comparable to an Excel Spreadsheet.

Columns in a DataFrame represents properties (such as a rating of a movie, the weight of a car, or length of an item) and rows in a DataFrame represents values as shown in Figure 2.12.

Figure 2.13: A Data frame example



userId	movieId	rating	timestamp
1	110	1.0	1425941529
1	147	4.5	1425942435
1	858	5.0	1425941523
1	1221	5.0	1425941546
1	1246	5.0	1425941556

It is possible to use them to manipulate, explore, and import the data. Additionally, SQL queries might be used within Spark syntax.

3. PROJECT DESIGN

3.1 MOTIVATION

The main motivation behind this thesis study is the curiosity and desire to improve the performance of Alternating Least Squares recommendation algorithm. It is a very popular algorithm for building recommendations. In this thesis study, ALS is used as one of the algorithms that are hybridized. The other algorithm used for hybridization is an algorithm that is created specifically to use the aspect of dissimilarity between users. In most of the recommender system algorithms, generally, the similarity between users is used. In this study, the idea of using an aspect that is generally not used acts as another motivator.

3.2 ENVIRONMENT SETUP

Apache Spark is one of the well-known and commonly used big data processing frameworks. It processes iterative algorithms by using in-memory computing technology. Compared to Hadoop's MapReduce technology, a broader range of capabilities is supported by Apache Spark (Dean and Ghemawat 2004, pp.137-150). Spark uses RDD as its programming block and it is the main reason Spark is more successful than MapReduce (Zaharia et al. 2011). RDDs are an unchangeable distributed group of objects and separated over several parts for computation on different nodes in a parallel fashion. Another fact that increases the effectiveness of Apache Spark is Lazy Evaluation strategy (Armbrust et al. 2015, pp.1383-1394). Apache Spark is selected as the Big Data Analytics framework for this project.

3.2.1 Spark Installation

There are different ways to use Apache Spark. It can be installed on a Windows or Linux machine, or it can be used in pre-prepared sandbox solutions such as Hortonworks Data Platform – HDP, MAPR Sandbox, etc.

There are four different deployment modes of Apache Spark:

i. Local
ii. Standalone
iii. Apache YARN
iv. Apache Mesos

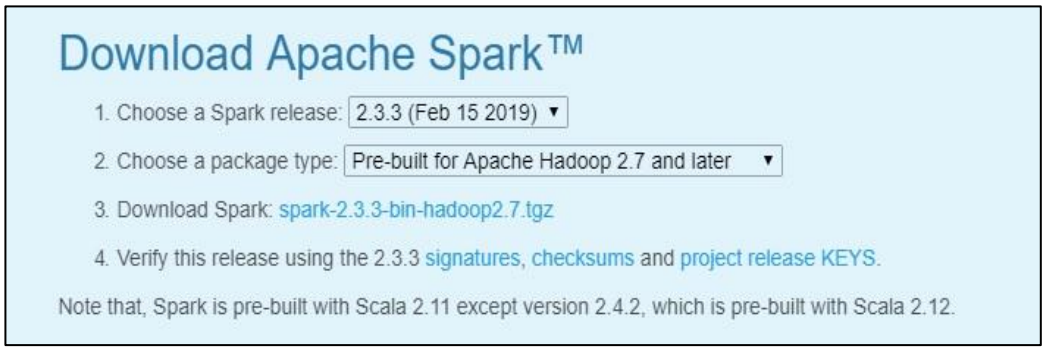
Spark may be set up with different cluster managers such as Apache Mesos, or Hadoop YARN. It can also be set up in local or standalone mode. In this project, the local deployment mode is used. In local deployment mode, the jobs in Spark are operated on a single machine and implemented in a parallel fashion by using multi-threading technology. There is a limitation about the parallelism in this type of deployment model. The parallelism is done along with the number of cores and processors on the machine. Spark is installed on a virtual machine which runs Windows 10 operating system. The virtual machine runs on top of a Microsoft Hyper-V Host. The main configuration of the virtual machine is listed below:

Operating System	: Windows 10
RAM	: 128 GB
CPU	: 16 Virtual Processors
Hard disk	: 90 GB

Below are the steps to successfully install Apache Spark to Windows 10 machine:

i. Download and install the Java Development Kit (JDK) 8+,
ii. Set environment variables for JAVA_HOME,
iii. Go to https://spark.apache.org, choose the proper version from the options shown in Figure 3.1, and download Apache Spark,

Figure 3.1: Apache Spark download options



iv. Extract the contents of the downloaded archive file to C:\Spark folder,

v. Configure C:\Spark\conf\log4j.properties.template file by setting *log4j.rootCategory=ERROR* and rename the file as log4j.properties,

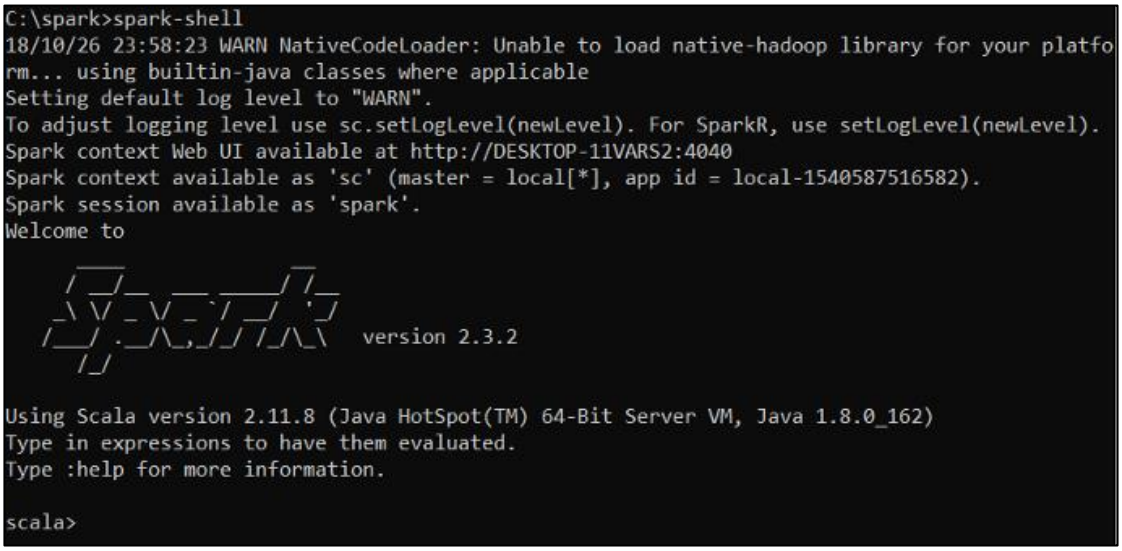
vi. Set environment variables for SPARK_HOME,

vii. Download winutils.exe to C:\Hadoop\bin folder

viii. Set environment variables for HADOOP_HOME

ix. Finally, open Command Prompt and type `spark-shell` to check if the installation is successful. Once successfully installed, the output of the `spark-shell` command should be similar to one in Figure 3.2.

Figure 3.2: Verification of Spark installation



x. By default Apache Spark is initialized using Scala programming language. If python is the desired programming language, type `pyspark –master local[2]` to initialize Apache Spark using PySpark as shown in Figure 3.3.

Figure 3.3: Pyspark initialization

In the command *`pyspark --master local[n]`*, *n* specifies how many worker threads will be run locally on Spark. The number of cores on the machine can be used as *n*.

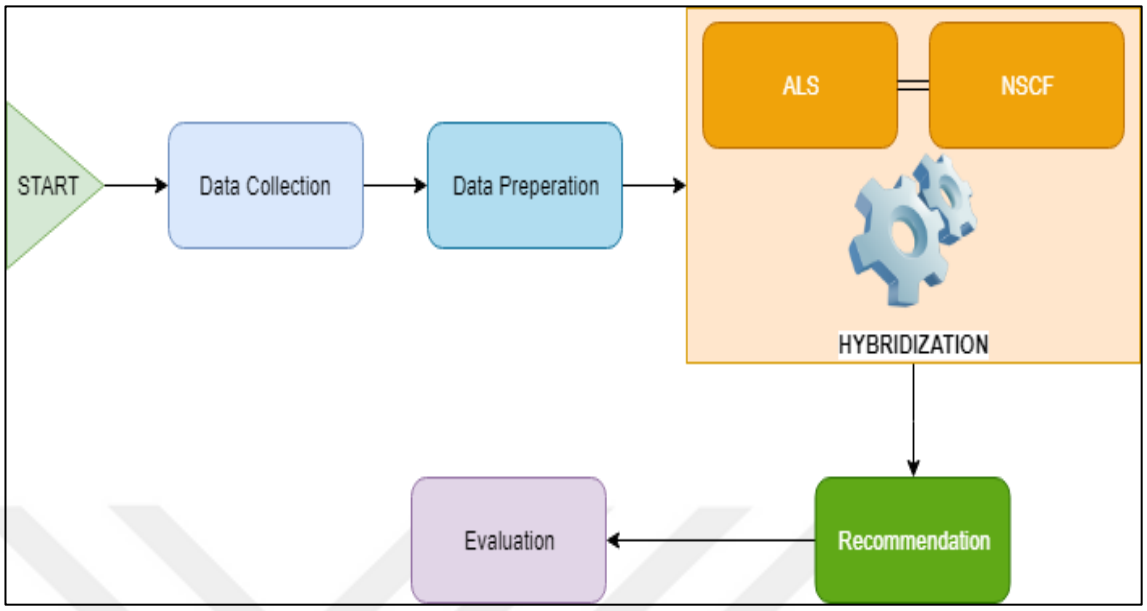
3.3 DATASET

In this thesis study, MovieLens dataset, which is a very popular, well-known, and widely used dataset in recommender system researches, was used. There are multiple variants of MovieLens dataset for research, education and development purposes. In this thesis study, MovieLens 20M dataset was used. Additionally, there are other versions such as 100K, 1M, and 10M samples. The dataset includes samples with a *userId* which specifies a particular user, a rating which is given to a movie by that user, the *movieId* which specifies a particular movie, and the timestamp which specifies the time the user has given the rating.

3.4 MODEL BUILDING

A hybrid approach is selected to be used in this thesis work. The main algorithms to be hybridized are ALS and a modified user-user collaborative filtering algorithm by using negative correlation. Hybridization was applied to the algorithms and cascade hybridization method was used to combine the results. The model design of the overall recommendation is shown in Figure 3.4.

Figure 3.4: Proposed model



Firstly, the raw data is examined and pre-processing is applied to the data since it is quite sparse. The users with less than 20 ratings should be filtered out in order to have a cleaner dataset to create the recommendation.

3.4.1 Alternating Least Squares Implementation

Once the dataset is collected and pre-processed for the implementation, ALS algorithm in Apache Spark's MLlib library is applied for the selected user and a set of recommendations are stored in a temporary recommendation list database.

3.4.2 User-User Collaborative Filtering by Negative Correlation Implementation

In the negative correlation implementation phase, the essence of the algorithm is to find *n* users that have the most dissimilar tastes with a particular user and try to recommend the movies lowly-rated by these dissimilar users to that particular user. The algorithm used for this purpose is called *Negative Similarity Collaborative Filtering – NSCF*. To create such a recommendation list, firstly, *n* number of users is calculated as the highly negatively correlated users to the particular user by using the Pearson Correlation

Coefficient. After finding n highly negatively correlated users, their *m* lowest ratings – rating ranging from 0.5 to 5 – are stored in the temporary recommendation list database as well.

Algorithm for Negative Similarity Collaborative Filtering - NSCF

Input : Number of users used as negative reference *n*,
User to recommend items to u_i,
User to be selected as negative reference u_j,
Pearson Correlation Coefficient for reference user r_{Uj}
List of all users *Users*,

Output : *m* items to be recommended

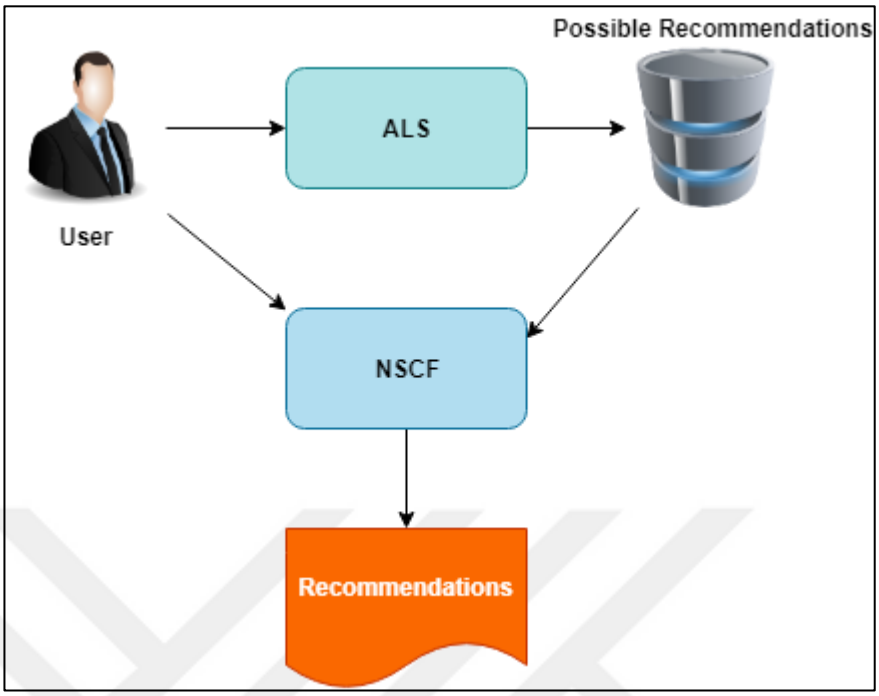
for each *user* ∈ *Users* **do**
  calculate r_{Uj} for the *user*
  user.rank ← *rank_based_on_*r_{Uj}
order_by_ascending
return top(*n*, *Users*)

3.4.3 Hybridization

Cascade hybridization method was used in the project as the hybridization method in order to increase the overall recommendation performance. In cascade hybridization technique, first method's recommendations are refined by the second method's recommendation. The first recommendation method results in a rough recommendations list. Then, that list is processed by the second recommendation method.

In the implementation of the hybridization, first of all, the ALS algorithm is applied for the selected user and a possible list of recommendations is stored in a temporary recommendations list database. Then the NSCF algorithm is applied for the selected user and a list of recommendations created. It is then compared to the list of recommendations created by the ALS algorithm. The movies that are listed in both of the recommendation lists are selected and the ultimate recommendation list is created. Then, that list is presented as the recommendations to the particular user.

Figure 3.5: Hybridization model



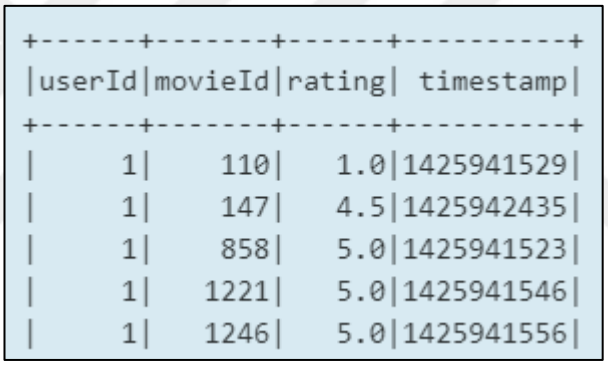
4. IMPLEMENTATION AND EVALUATION

4.1 DATA ANALYSIS

There are various MovieLens datasets available for research, education, and development purposes. In this project, MovieLens 20M Dataset was used. The dataset contains the ratings of the users for the movies. The data is collected over 20 years -from 1995 to 2015. The dataset was created on October 17, 2016.

MovieLens 20M Dataset contains six files. In this study, *ratings.csv* file was used to create the recommender system. The format of this file is shown in Figure 4.1.

Figure 4.1: ratings.csv file



userId	movieId	rating	timestamp
1	110	1.0	1425941529
1	147	4.5	1425942435
1	858	5.0	1425941523
1	1221	5.0	1425941546
1	1246	5.0	1425941556

The file is ordered first by userId and then movieId. The ratings are given from 0.5 to 5 with 0.5 increments.

4.1.1 Data Exploration

The ratings.csv dataset includes 20000263 ratings. That's why it is named 20M dataset. The ratings are given from 138493 distinct users to 26744 distinct movies as shown in Table 4.1. Each user and each movie have a different number of ratings.

Table 4.1: Key information for ratings.csv

Property	Count
Users	138493
Movies	26744
Ratings	20000263

The average rating for the movies is approximately 3.53 and the standard deviation is approximately 1.05. The ratings range from 0.5 to 5 as shown in Table 4.2.

Table 4.2: Key statistics for ratings

ratings			
Mean	Std deviation	min	max
3,5255	1.0519	0.5	5

4.1.2 Data Preparation

There are no null values in the dataset as shown in Figure 4.2. Every single user has rated a movie, and they each have at least 20 ratings. Since each user has 20 ratings and there are no null values in the dataset, no cleaning process is applied to the dataset. Some movies are rated only once. Those movies might have been removed from the dataset to have a less sparse dataset, but to have the possibility to recommend movies from the long tail, they are kept intact.

Figure 4.2: Null values in the dataset

```
# Count null value
from pyspark.sql.functions import col,lit

rows = df.count()
summary = df.describe().filter(col("summary") == "count")
summary.select(*((lit(rows)-col(c)).alias(c) for c in df.columns)).show()
```

```
+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
|   0.0|    0.0|   0.0|      0.0|
+------+-------+------+---------+
```

4.2 ALS IMPLEMENTATION

Alternating Least Squares algorithm can be used in Apache Spark's Machine Learning library – called *spark.ml*. There are various parameters used in spark.ml. The parameters used in ALS implementation are listed below:

i. *numBlocks* specifies the amount of chunks utilized to parallelize computation,
ii. *maxIter* specifies the amount of repetitions the program will run at a maximum,
iii. *regParam* is ALS's regularization parameter,
iv. *rank* specifies the amount of latent factors in the model,
v. *nonnegative* specifies the usage of nonnegative restraints for least squares

If the parameter is not specified in the code, default values for the parameters are used. Spark lets *coldStartStrategy* criterion to be defined as 'drop' for dropping rows in the dataset that include NaN values.

In ALS implementation, first of all, the required libraries that are imported. Then the data is assigned to a data frame. The data is then randomly split into train and test datasets. The training dataset corresponds to 80% of the raw dataset and the test dataset corresponds to 20% of the raw dataset. ALS algorithm is applied to train dataset as shown in Figure 4.3.

Figure 4.3: ALS implementation on the dataset

```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator

# Load ratings data to dataframe
df = spark.read.csv('C:/data/ml-20m/ratings.csv', inferSchema=True, header=True)

# Split data into train and test data
train, test = df.randomSplit([0.8,0.2])

# Build a model using ALS on the training data
# Set coldstartStrategy to 'drop' to ensure no NaN values are evaluated
als = ALS(maxIter=10, regParam=0.1, rank=8, nonnegative=True, coldStartStrategy="drop", \
          userCol='userId', itemCol='movieId', ratingCol='rating')
model = als.fit(train)
```

After the ALS algorithm is applied to the training dataset, we can get predictions for the test dataset as shown in Figure 4.4.

Figure 4.4: Predictions for the test dataset

```
# Show predictions on the test data
predictions = model.transform(test)
predictions.show()
```

```
+------+-------+------+---------+----------+
|userId|movieId|rating|timestamp|prediction|
+------+-------+------+---------+----------+
| 22684|    148|   4.0|832057800|  2.935126|
| 92852|    148|   3.0|839813031| 2.5590916|
|123246|    148|   3.0|833017056| 3.1063094|
|  5585|    148|   3.0|833940677| 3.4032044|
| 46146|    148|   2.0|839629075| 1.9105072|
| 46944|    148|   2.0|839965214| 2.8752956|
| 54726|    148|   5.0|832703670| 3.4011936|
| 90757|    148|   3.0|970000570| 3.2137938|
| 75781|    148|   3.0|895230335|  2.892173|
| 77165|    148|   3.0|840699559| 2.8522413|
| 59570|    148|   3.0|835559154| 2.8339372|
|  3673|    148|   2.0|901681963| 2.4727173|
|111523|    148|   2.0|942665835| 3.2763891|
| 61663|    148|   2.0|874577512|  3.083672|
|132268|    148|   2.0|915406133| 2.1895466|
|   603|    148|   2.0|838308516|  3.080516|
| 36723|    148|   1.0|833673768|  2.473158|
| 61815|    148|   3.0|833165538| 3.5821662|
| 37331|    148|   3.0|944952722| 2.6184487|
| 10434|    148|   3.0|837033792|  2.623882|
+------+-------+------+---------+----------+
only showing top 20 rows
```

The *predictions* data frame is where ALS predictions for all users are stored. This data frame is used as a reference during the hybridization process.

4.3 NEGATIVE SIMILARITY IMPLEMENTATION

In the Negative Similarity Collaborative Filtering implementation, *movies.csv* and *ratings.csv* files are used. *movies.csv* file is used to extract the movie title information and incorporate it to the final data frame. These two files are loaded into respective data frames and unnecessary columns are removed from both of the data frames. Then these data frames are merged, column order is changed so that the columns are listed in this

order: *userId, movieId, title, rating.* Finally, the data frame is ordered by *userId* as shown in Figure 4.5

Figure 4.5: Data frame preparation

```
import pandas as pd

# Load movies.csv and ratings.csv
movies = pd.read_csv('C:/data/ml-20m/movies.csv')
ratings = pd.read_csv('C:/data/ml-20m/ratings.csv')

# Select only needed columns
movies = movies.loc[:,["movieId","title"]]
ratings = ratings.loc[:,["userId","movieId","rating"]]

# Merge two dataframes to have movie titles in the dataframe
data = pd.merge(movies,ratings)

# Change column orders to this order: userId, movieId, title, rating
cols = ['userId', 'movieId', 'title', 'rating']
data = data[cols]

# Sort dataframe by userId and movieId
data = data.sort_values(['userId','movieId'])

# Show the head of the dataframe
data.head()
```

	userId	movieId	title	rating
49695	1	2	Jumanji (1995)	3.5
346885	1	29	City of Lost Children, The (Cité des enfants p...	3.5
366121	1	32	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	3.5
526043	1	47	Seven (a.k.a. Se7en) (1995)	3.5
582574	1	50	Usual Suspects, The (1995)	3.5

Next, the user-item matrix is created. This can be achieved by using *pivot_table* function. With this function, rows are set as movies, columns are set as users, and the ratings are set as the values as shown in Figure 4.6.

Figure 4.6: User-Item matrix

```
pivot_table = data.pivot_table(index = ["title"], columns = ["userId"], values = ["rating"])
pivot_table.head()
```

	rating													
userId	1	2	3	4	5	6	7	8	9	10	...	6734	6735	6736
title														
"Great Performances" Cats (1998)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
$5 a Day (2008)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
'71 (2014)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
'Hellboy': The Seeds of Creation (2004)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN
'Neath the Arizona Skies (1934)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN

To calculate the Negative Similarity a random user is selected and this user is sent to *calculate_similarity* function as a parameter. The *calculate_similarity* function returns a list which includes the correlation between the selected user and other users ordered by from the most negative to the most positive.

Then *Top-N* users are selected from the list and fed into the *create_recommendation_list* function as a parameter together with how many items, denoted by *m*, will be extracted from the users as shown in Figure 4.7. The function *create_recommendation_list* creates a recommendation list by taking the average of each extracted movie and sorting the list from low to high.

Figure 4.7: Calculating negative similarity

```
# Select a random user
user = pivot_table.iloc[:,1036]

# Calculate similarity between selected user and other users
similar_users = calculate_similarity(user)

# Define n and m
n = 10 # how many similar users will be used
m = 10 # how many movies will be extracted from each user

# Create recommendation list
recommendation_list = create_recommendation_list(n,m)
```

4.4 HYBRIDIZATION

Hybridization process is done once the ALS algorithm is applied to the dataset. After applying ALS, a random user is selected to start the hybridization process. A recommendation list for the selected user is created by the ALS algorithm as shown in Figure 4.8.

Figure 4.8: ALS recommendation list

```
# A list of movies that are recommended to user 1036 by ALS algorithm
user_suggest = test.filter(train['userId'] == 1036).select(['movieId','userId'])

user_offer = model.transform(user_suggest)
user_offer.orderBy('prediction', ascending=False).show()
```

```
+-------+------+----------+
|movieId|userId|prediction|
+-------+------+----------+
|   2959|  1036| 4.7422347|
|     50|  1036| 4.6608624|
|  58559|  1036|  4.653611|
|    296|  1036| 4.5141206|
|   3275|  1036|  4.488466|
|   1089|  1036| 4.3830347|
|   6874|  1036| 4.3385005|
|   8784|  1036| 4.3242497|
|  58998|  1036| 4.1208653|
|   3476|  1036|  4.053483|
|   3676|  1036| 3.5286014|
|   1779|  1036| 3.3257556|
+-------+------+----------+
```

Then, the NSCF algorithm is applied to the same user to create the second recommendation list. Once two recommendation lists are available at hand. Selection of recommended movies that are in common in both of the list is done. The selected list is put into another data frame and ordered as the highest-rated recommended movie is listed on the top. The list is then recommended to the selected user.

4.5 EVALUATION

The evaluation of the proposed hybrid recommender system is done by measuring the overall performance of the hybrid recommender system and the performances of the ALS algorithm and NSCF algorithm which are part of the hybridization. In order to do that, first of all, individual algorithms' performances are measured during their respective implementation. Once the hybridization is done, the overall performance of the proposed hybrid recommender system is measured and it is then compared with the individual algorithms' performances in turn.

First of all, Root Mean Square Error – RMSE metric is used to calculate and compare the performances of the hybrid recommender system and the individual recommender systems. Later on, Receiver Operating Characteristic (ROC) and Area Under ROC are calculated for the hybrid recommender system. Finally, the precision and recall values are calculated for the hybrid recommender system.

Evaluation of the ALS algorithm is measured after the ALS model is created and the predictions are made on the test dataset as shown in Figure 4.9. The RMSE of the ALS algorithm is approximately 0.8158.

Figure 4.9: Evaluation of the ALS algorithm

```
# Evaluate ALS algorithm by calculating the RMSE on the test dataset
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName='rmse', labelCol='rating',
                                predictionCol='prediction')
rmse = evaluator.evaluate(predictions)
print("Root mean squared error :" + str(rmse))
```

```
Root mean squared error :0.815794981237273
```

Evaluation of the NSCF algorithm is measured after the NSCF model is created and the predictions are made on the test dataset. In order to evaluate the performance of the NSCF algorithm, *evaluate_rmse* function is created. This function takes the recommendation list

as a parameter and returns the calculated RMSE value as shown in Figure 4.10. The RMSE of the NSCF algorithm is approximately 3.158.

Figure 4.10: Evaluation of the NSCF algorithm

```
# Evaluate NSCF algorithm by calculating the RMSE
rmse = evaluate_rmse(recommendation_list)

print("Root mean squared error :" + str(rmse))
```

```
Root mean squared error : 3.1579546374729742
```

The hybridized model's evaluation is calculated by the *evaluate_hybrid_rmse* function. This function takes the hybridized recommendation list as a parameter and returns the RMSE value of the hybridized recommendation as shown in Figure 4.11. The RMSE of the hybridized model is approximately 0.7912.

Figure 4.11: Evaluation of the Hybridized model

```
# Evaluate hybridized model by calculating the RMSE
rmse = evaluate_hybrid_rmse(hybrid_recommendation_list)

print("Root mean squared error :" + str(rmse))
```

```
Root mean squared error : 0.7947820387921647
```

The RMSE of the hybridized model is slightly better than the RMSE of the ALS algorithm. When it is compared to the RMSE of the NSCF algorithm, the difference is significant. It can be concluded that the hybridized model slightly improves the performance of the ALS algorithm applied alone.

Another evaluation metric used for the hybrid recommender system is the Receiver Operating Characteristic (ROC). ROC curve is a frequently used metric for evaluating the performance. This is calculated by using the *BinaryClassificationEvaluator* function

of the Apache Spark's *pyspark.ml.evaluation* library as shown in Figure 4.12. ROC curve for the hybrid recommender system is shown in Figure 4.13. It is around 0.885 and can be considered as good.

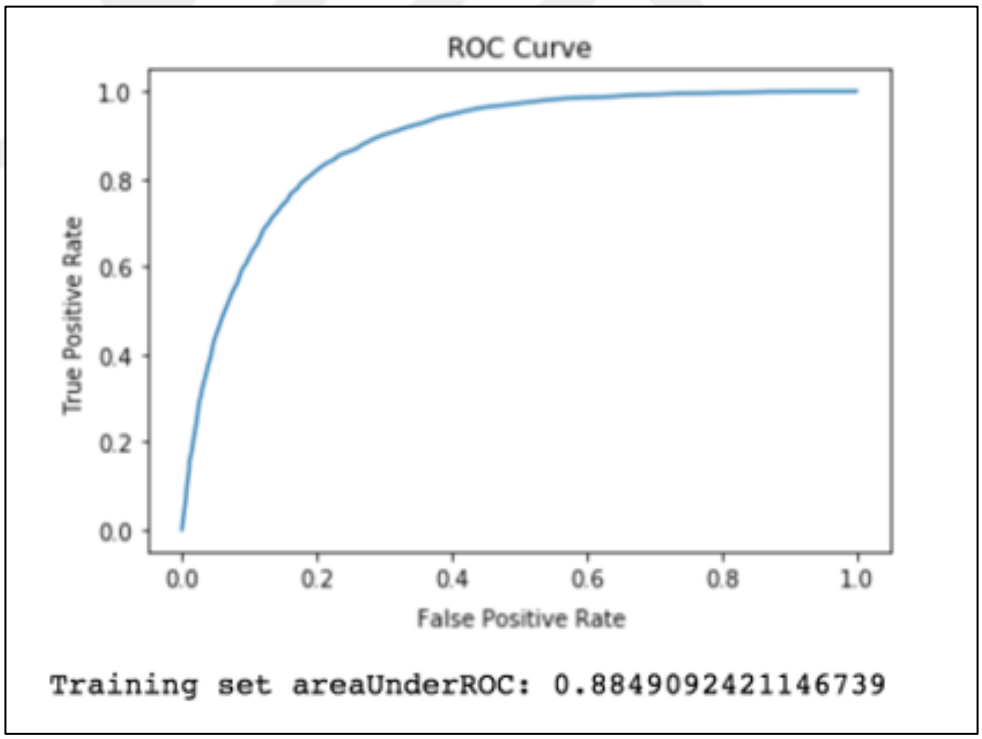
Figure 4.12: Calculating the ROC

```
# Evaluate the model by calculating Area under ROC
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print('Test Area Under ROC', evaluator.evaluate(hybrid_recommendation_predictions))
```

```
Test Area Under ROC 0.8849092421146739
```

Figure 4.13: Area under the ROC Curve



The precision score is used as the final evaluation for the hybrid recommender system. In order to calculate the precision, top 10 recommendations for the users in the test group are used. The precision score is calculated as shown in Equation 5.1.

$$Precision = \frac{\text{True Positive}}{True\ Positive + False\ Positive} \quad (5.1)$$

The precision score represents the recommender system's ability to create relevant predictions. In this study, if the user's rating for an item is greater than 3.5, then that item is considered relevant to the user and if the rating is smaller than 3.5, then that item is considered irrelevant to the user. In Equation 5.1, True positive represents the items that are recommended to the user and actually relevant to the user. False-positive represents the items that are recommended to the user but actually irrelevant to the user. The precision score for the hybrid recommender system is **0.7635**. It is the average of precision values for all the users.

5. DISCUSSION

This thesis study has provided a slight improvement in performance when compared to the Alternating Least Squares algorithm. This is mainly because it is not easy and feasible to improve a recommender system that performs well by a wide margin.

The algorithms used in the study for hybridization are variations of the collaborative filtering algorithm. Therefore, they are inclined to face the item cold start and user cold start problems. The cold start problem is one of the drawbacks of the collaborative filtering algorithm. To eliminate the item cold-start problem, content-based filtering algorithm might be used as the third algorithm to be hybridized in the model. This would provide results with better precision but require more time to execute since a third algorithm has to be applied to the model.

Additionally, to increase the efficiency of the recommender system, the effect of time over a long period might be added to the system as another metric to be supplied. The main idea behind this approach is the fact that people's interest might change or deviate by time, age, marital status, etc. Individual user's ratings to the movies might be split into smaller chunks, such as *n* ratings in a row. Then the features of movies rated like genre, cast, director, etc. are taken into account in order to create a pattern. This pattern might be used as a reference to find similar patterns of other users' and create recommendations based on these patterns.

6. CONCLUSION

This thesis study implements a hybrid recommender system using Apache Spark. The algorithms used in building the hybrid recommender system, including Alternating Least Squares and Negative Similarity Collaborative Filtering – a modified version of user-user based collaborative filtering, are examined in this thesis study.

Hybridization is used to increase the efficiency of the overall recommender system by using the advantages or strengths of the algorithms hybridized. They also try to ameliorate the disadvantages of the algorithms when used alone.

The hybridized algorithms are both flavors of Collaborative Filtering. Inherently, the implementation of collaborative filtering is a very expensive process because its model has to be updated whenever a new user preference comes in. Thus; having an analytics framework like Apache Spark for highly scaled data, helps to reduce the time for processing data, building models, and providing the results. Apache Spark's MLlib library ensures data analytics for highly scaled data by providing a plenty set of methods. ALS implementation of the MLlib library provides good results when used in collaborative filtering. Negative Similarity Collaborative filtering - a modified version of collaborative filtering – tries to provide different metrics to increase the overall efficiency of the recommender system.

The evaluation of the hybridized recommender system is compared with the individual algorithms used in hybridization. It is concluded that hybridization improves the performance slightly when compared to the ALS algorithm applied alone, while it improves the performance greatly when compared to the NSCF algorithm applied alone.

The provided hybrid recommender system is prone to face the cold start problem which is one of the weaknesses of collaborative filtering. This is mainly because the methods used for hybridization are flavors of collaborative filtering technique. In order to further improve the performance of the recommender system, content-based filtering technique

might be added to the system as the third algorithm. This will cost in longer execution time of the system though.

Another algorithm for calculating the effects of time on people's preferences might be used in the hybrid recommender system to reach an additional improvement. This can be achieved by finding the patterns such as watching and rating a particular genre in the frequency of a user's rating to the items.

REFERENCES

Books

Jannah D., Zanker M., Felfernig A. & Friedrich G., 2011. Recommender systems an introduction. New York: Cambridge University Press

Ricci F., Rokach L. & Shapira B., 2011. Recommender systems handbook. Second edition. New York: Springer Science+Business Media

Aggarwal C., 2016. Recommender systems the textbook. New York: Springer Science+Business Media

Sammut C. & Webb G., 2010. Encyclopedia of Machine Learning. Boston: Springer Science+Business Media.

Schafer JB, Frankowski D, Herlocker J, Sen S., 2007. Collaborative filtering recommender systems. In: Brusilovsky P, Kobsa A, NejdlW, editors. The Adaptive Web, LNCS 4321. BerlinHeidelberg (Germany): Springer pp. 291-324.

Burke R., 2007. Web Recommender Systems. In: Brusilovsky P, Kobsa A, Nejdl W, editors. The Adaptive Web, LNCS 4321. BerlinHeidelberg (Germany): Springer, pp. 377-408.

Zaharia M., Chambers B., 2018. Spark: The Definitive Guide. O'Reilly Media.

Bai X., Wu J., Wang H., Zhang M., Zhang J., Fu Y., Rui X., Yin W., Dong J., 2012. Recommendation Algorithms for Implicit Information. In: Service Science, Management, and Engineering. Academic Press, pp. 77-94.

Periodicals

Burke R., 2002. Hybrid recommender systems: survey and experiments. *User Modeling and User-Adapted Interaction.* **12** (4), pp. 331-370.

Bobadilla J, Ortega F, Hernando A, Gutierrez A., 2013. Recommender systems survey. Knowledge-Based Systems. 46, pp. 109-132.

Resnick P., & Varian H., 1997. Recommender systems. Communications of the ACM. **40** (3), pp. 56-58.

Adomavicius G., Tuzhilin A., 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering.* **17** (6), pp. 734-749.

Isinkaye F., Folajimi Y., Ojokoh B., 2015. Recommendation systems: Principles, methods, and evaluation. *Egyptian Informatics Journal.* **16** (3), pp. 261-273.

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

Liang, X., Xia Z., Pang L., Zhang L., Zhang H., 2016. Measure prediction capability of data for collaborative filtering. *Knowledge and Information Systems.* **49** (3), pp. 975-1004.

Park D., Kim H., Choi I., Kim J., 2012. A literature review and classification of recommender systems research. *Expert Systems with Applications.* **39** (11), pp. 10059-10072.

Su X., Khoshgoftaar T., 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence.* **2009**, pp. 4-19.

Zhang T., Iyengar V., 2002. Recommender Systems Using Linear Classifiers. *The Journal of Machine Learning Research.* **2** pp. 313-334.

Adomavicius G., Zhang J., 2012. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems.* **3** (1).

Smyth B., Cotter P., 2000. A personalized TV listings service for the digital TV age. *Knowledge-Based Systems*. **13** (2-3), pp. 53-59.

Burke R., Hammond K., Young B., 1997. The FindMe Approach to Assisted Browsing. *IEEE Expert: Intelligent Systems and Their Applications.* **2** (4), pp.32-40.

Pazzani M., 1999. A Framework for Collaborative, Content-based and Demographic Filtering. *Artificial Intelligence Review.* **5** (6), pp. 393-408.

Balabanovic M., Shoham Y., 1997. Fab: content-based, collaborative recommendation. *Communications of ACM Magazine.* **40** (3), pp. 66-72.

Friedman N., Geiger D., Goldszmidt M., 1997. Bayesian Network Classifiers. *Machine Learning.* **29** (2-3), pp. 131-163.

Goldberg K., Roeder T., Gupta D., Perkins C., 2001. Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval.* **4** (2), pp.133-151.

Papagelis M., Plexousakis D., 2005. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. *Engineering Applications of Artificial Intelligence.* **18** (7), pp. 781-789.

Albadvi A., Shahbazi M., 2009. A Hybrid recommendation technique based on product category attributes. *Expert Systems with Applications.* **36** (9), pp. 11480-11488.

Cho Y., Kim J., 2004. Application of Web usage mining and product taxonomy to collaborative recommendations in e-commerce. *Expert Systems with Applications.* **26** (2), pp. 233-246.

Schafer J., Konstan J., Riedl J., 2001. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*. **5** (1/2), pp.115–153.

Thorson E., 2008. Changing patterns of news consumption and participation. *Information, Communication & Society*. **11** (4), pp.473–489.

Koren Y., Bell R., Volinsky C., 2009. Matrix factorization techniques for recommender systems. *Computer*. **42** (8), pp. 30–37.

Billsus D., Pazzani M., 2000. User Modeling for Adaptive News Access. *User Modeling and User-Adapted Interaction*. **10** (2-3), pp. 147-180.

Other Publications

Breese J., Heckerman D., & Kadie C, 1998. Empirical analysis of predictive algorithms for collaborative filtering. *UAI'98 Proceedings of the Fourteenth Conference on Uncertainty in artificial intelligence*, pp. 43-52. July 24 - 26, 1998 Madison, Wisconsin, USA.

Zhao Z., Shang M., 2010. User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop. *WKDD '10 Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining*, pp. 478-481. January 09-10, 2010 Washington, USA.

Hahsler M., 2014. recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.1-5. URL: http://CRAN.R-project.org/package=recommenderlab

Melville P., Mooney R., Nagarajan R., 2002. Content-boosted collaborative filtering for improved recommendations. *Eighteenth national conference on Artificial intelligence*, pp. 187-192. July 28 - August 01, 2002 Edmonton, Alberta, Canada.

Shyong K, Frankowski D, Riedl J., 2006. Do you trust your recommendations? An exploration of security and privacy issues in recommender systems. *Proceedings of Emerging Trends in Information and Communication Security*, pp.14-29. June 6-9, Freiburg, Germany.

Stern D., Herbrich R., Graepel T., 2009. Matchbox: large scale online bayesian recommendations. *Proceedings of the 18th International Conference on World Wide Web*, pp. 111-120. April 20-24, Madrid, Spain.

Claypool M., Gokhale A., Miranda T., Murnikov P., Netes D., Sartin M., 1999. Combining Content-Based and Collaborative Filters in an Online Newspaper. *Proceedings Of ACM SIGIR Workshop On Recommender Systems: Algorithms And Evaluation.* Berkeley, California, USA.

Billsus D., Pazzani M., 1999. A hybrid user model for news story classification. *Proceedings of the seventh international conference on user modeling*, pp.99-108. Banff, Canada.

Wasfi A., 1999. Collecting user access patterns for building user-profiles and collaborative filtering. *Proceedings of the 1999 international conference on intelligent user*, p. 57–64. Redondo Beach, CA, USA.

Basu C., Hirsh H., Cohen W., 1998. Recommendation as classification: using social and content-based information in recommendation. *Proceedings of the 15th national conference on artificial intelligence*, pp. 714-720. Madison, Wisconsin, USA.

Mooney R., Roy L., 2000. Content-based book recommending using learning for text categorization. *Proceedings of the fifth ACM conference on Digital libraries,* pp. 195-204. June 02-07, San Antonio, Texas, USA.

Cotter P., Smyth B., 2000. PTV: Intelligent Personalised TV Guides. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 957-964. July 30 – August 03.

Sarwar B., Karypis G., Konstan J., Reidl J., 2001. Item-based collaborative filtering recommendation algorithms. *Proceedings of the10th international conference on World Wide Web, ACM*, pp. 285-295. May 01-05, Hong Kong, China.

Dean J., Ghemawat S., 2004. Map-reduce: Simplified Data Processing on Large Clusters. *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation*, pp.137-150. December 06-08, San Francisco, CA, USA.

Zaharia S., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M., Shenker S., Stoica I., 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation.* April 25-27, San Jose, CA, USA.

Armbrust M., Xin R., Lian C., Huai Y., Liu D., Bradley J., Meng X., Kaftan T., Franklin M., Ghodsi A., Zaharia M., 2015. Spark SQL: Relational Data Processing in Spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 1383-1394. May 31 - June 04, Melbourne, Victoria, Australia.

https://www.iso.org/standard/11833.html

http://entertainment.time.com/2012/07/10/go-ahead-binge-watch-that-tv-show/