RECOMMENDATION SYSTEM USING FACTORIZATION MODEL AND MAPREDUCE FRAMEWORK

by

Nabin Giri

An Abstract
of a thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science
in the School of Computer Science and Mathematics
University of Central Missouri

May, 2020

ABSTRACT

by

Nabin Giri

Recommendation System is a subclass of information filtering system which helps us to predict the rating for an item based on the information available of user and items itself. Based on those ratings, we provide personalized recommendation to user about the items. Recommendation System helps to increase the decision making capacity for users and increases the product sales for an ecommerce website. However, building a recommendation system has two major challenges, one being that there are many algorithms we want to use and each one of them have their own advantages and drawbacks. Secondly, the system has to deal with large user and item data. In this thesis, we define a model which is capable of processing different collaborative filtering algorithm simultaneously. Furthermore, in order to handle large user and item data, we use MapReduce framework. We have performed experiments on the model and interesting results on the relative performance measures are shown at the end of the thesis.

RECOMMENDATION SYSTEM USING FACTORIZATION MODEL AND MAPREDUCE FRAMEWORK

by

Nabin Giri

A Thesis
presented in partial fulfillment
of the requirements for the degree of
Masters of Science
in the School of Computer Science and Mathematics
University of Central Missouri

May, 2020

RECOMMENDATION SYSTEM USING FACTORIZATION MODEL AND MAPREDUCE FRAMEWORK

by

Nabin Giri

May, 2020

UNIVERSITY OF CENTRAL MISSOURI

WARRENSBURG, MISSOURI

ACKNOWLEDGEMENTS

I would like to begin by thanking my advisor Dr. Yui Man Lui for introducing me to collaborative filtering, for many helpful sessions and for numerous constructive comments about this thesis itself. I offer my gratitude to co-advisor and graduate program advisor Dr. Mahmoud Yousef for suggestions and guidance on this thesis and also on many other occasions. I would also like to thank my second co-advisor Dr. Dabin Ding for helpful insights and time to review this thesis. Empirical research into recommendation system is not possible without large computation power. My sincerest thanks and appreciation to Graduate Education and Research Office for continued support and funds on using Amazon Cloud services. I would also like to extend my deepest gratitude to the members of James C. Kirkpatrick Library for unwavering support throughout my academic years.

TABLE OF CONTENTS

Page

LIST OF TABLES xi

LIST OF FIGURES xii

CHAPTER 1: INTRODUCTION 1

CHAPTER 2: LITERATURE REVIEW 9

2.1 Neighborhood Methods 9

2.2 Latent Factor Models 12

2.3 Latent Factor Meets Neighborhood 14

2.4 Handling Large Scale Data 14

CHAPTER 3: FACTORIZATION MODEL 16

3.1 Approach 16

3.1.1 Notation 17

3.2 Data Normalization 17

3.3 Matrix Factorization Model 19

3.3.0.1 UV Decomposition 22

3.3.0.2 Initialization and Formulation 23

3.4 Neighborhood Information 25

3.5 Model Formulation 27

3.5.0.1 Implementation 30

3.6 Integrated Model 32

3.6.0.1 Linear Regression 33

3.6.0.2 Feature-Based Factorization Model 36

3.7 Learning Algorithm 39

3.7.0.1 Cost Function 39

3.7.0.2 Model Learning 41

3.8 Update Rules 42

3.8.0.1 Generating Update Rule for User Factor 43

3.8.0.2 Generating Update Rule for Item Factor 44

3.8.0.3 Generating Update Rule for Global Bias 45

3.8.0.4 Generating Update Rule for User Bias 46

3.8.0.5 Generating Update Rule for Item Bias 47

3.8.0.6 Generating Update Rule for Neighborhood Parameter 48

CHAPTER 4: MAPREDUCE 50

4.1 Architecture 50

4.1.1 Mapper 51

4.1.2 Reducer 52

4.1.3 Combiner 52

4.1.4 Shuffle and Sort 54

4.1.4.1 Shuffle and Sort - A Map View 55

4.1.4.2 Shuffle and Sort - A Reduce View 55
4.2 mrjob 56
4.2.1 Factorization Model using mrjob 57
4.2.1.1 Architecture 58
4.2.1.2 Data Format 60
4.2.1.3 Mapping Phase 61
4.2.1.4 Shuffle and Sort : Map Phase 64
4.2.1.5 Combiner Phase 64
4.2.1.6 Shuffle and Sort : Reduce Phase 65
4.2.1.7 Reduce Phase 65
4.2.2 Running mrjob 66
4.2.2.1 Workflow 67
4.2.2.2 Running Locally 70
4.2.2.3 Running on a Cluster using Amazon EMR . . . 72

CHAPTER 5: EXPERIMENTS 76
5.1 Research Questions 76
5.2 Evaluation Criteria 77
5.3 Experimental Design and Analysis 78
5.3.1 Dataset 78
5.3.2 Experimental Procedure 79
5.3.3 Experiment : ML100K 81
5.3.4 Experiment : ML1M 83
5.3.5 Experiment : ML10M 85

5.3.6 Experiment : ML25M 88
5.3.7 Overall Performance 90
5.4 Results and Discussion 91

CHAPTER 6: CONCLUSION AND FUTURE WORKS 93

REFERENCES 96

APPENDIX 106
A. HUMAN SUBJECT APPROVAL 106

LIST OF TABLES

Table Page

3.1 A fabricated data which consists of four users for five movies . . . 21

3.2 Similarity of item Titanic to other items rated by user Charlie . . 31

5.1 MovieLens Datasets 79

5.2 RMSE of 100K five-fold 82

5.3 Configuration of EMR Instances for ML100K, ML1M, and ML10M 82

5.4 Processing time for 100K five-fold in EMR 83

5.5 RMSE of 1M five-fold 85

5.6 Processing time for ML1M five-fold in EMR 85

5.7 RMSE of 10M five-fold 86

5.8 Processing time for ML10M five-fold in EMR 88

5.9 RMSE of 25M five-fold 88

5.10 Configuration of EMR Instances for ML25M 90

5.11 Processing time for ML25M five-fold in EMR 90

5.12 Average RMSE of MovieLens Dataset 91

LIST OF FIGURES

Figure Page

3.1 Features of user James and Anne in feature space. 20

4.1 MapReduce Architecture 51

4.2 Shuffle and Sort Architecture 54

4.3 Workflow of mrjob Program 59

4.4 Schematic of Five-Fold Cross-Validation 60

4.5 Conversion of Raw Data to Feature Format Data 61

4.6 Workflow of Factorization Model with MapReduce Framework . . 66

4.7 EMR 1 74

4.8 EMR 2 75

5.1 RMSE by Epoch for ML100K five-fold 82

5.2 Processing time for ML100K five-fold in EMR 83

5.3 RMSE By Epoch for ML1M five-fold 84

5.4 Processing time for ML1M five-fold in EMR 85

5.5 RMSE By Epoch for ML10M five-fold 86

5.6 Processing time for ML10M five-fold in EMR 87

5.7 RMSE By Epoch for ML25M five-fold 89

5.8 Processing time for ML25M five-fold in EMR 90

5.9 Performance measure of MovieLens Dataset on EMR 91

CHAPTER 1
INTRODUCTION

The problem of information overload was first identified in 1982 by ACM President's Letter titled Electronic Junk by Peter J. Denning (Denning, 1982). Denning argued that with the shift of manual business processes to automation, we should focus more on the process of receiving and controlling the information so that it reaches the concerned person and not overwhelm them with unnecessary information. Subsequently more than a decade after Denning's letter, Web 2.0 (International and Media, 2004) was developed with a theme of "The Web as Platform" to connect all the businesses, companies and institutions in one arena to debate and discuss the issues and strategies driving the internet economy. Today, Web 2.0 allows internet users to share their opinion and feedback about almost everything. Since the development of Web 2.0, the internet has seen a massive surge of e-commerce and social networking sites such as Amazon, Facebook, Instagram, YouTube, Twitter and Netflix. These sites have become a part of people's daily life. In fact, users spend 140 million hours watching Netflix (Netflix, 2017) videos and over two billion users together spend one billion hours watching YouTube (YouTube, 2020) videos daily. 300 hours (Satista,

2019) of videos are uploaded to YouTube every minute. YouTube is only the tip of the iceberg, the explosive rate of increase in the new content has caused an inevitable information overload problem. As argued by Denning, we now need a remedy to tackle the information overload problem where the users are drowning themselves with the abundance of choices they have, and not everyone fits the mold.

An information filtering (IF) system is one of the methods that is evolving to manage and ease the information overload problem. Information Filtering was first introduced by Luhun (Luhn, 1958) and later named "Filtering" by Denning in 1975. Currently, these techniques are plentifully used in different applications like voice recognition, classification (i.e. to identify if an electronic mail is a spam or not), web browsers (i.e., web filters that block non-valuable information (e.g. advertisement)) from the web page. IF system works by removing the redundant or unwanted information using a set of predefined rules prior to presenting to a user. In order to do so, the user's profile is compared to a set of rules which then identifies the preference of an item by the user. These sets of rules are generated using two main approaches namely, a content-based approach and a collaborative filtering approach.

A content-based approach, which is also referred to as cognitive filtering is based on a characterization of the content of the information. A user's information needs are provided in terms of keywords as filtering rules. The contents of the available information are then matched against this list of keywords provided by the user in order to determine his/her preference. The preferred keywords or opinions are identified using two methods: explicit feedback and implicit feedback. This paper focuses on the domain of movies. In the movie domain we introduce explicit feedback as a rating provided by a user to a movie based on his/her preference of the movie. A rating is any value between one to five, where five is the highly preferred and, one being the least preferred movie by the user. Implicit feedbacks are the indirectly extracted information from user's activities, such as the movie rental history of the user, or the type and length of page views on the web. In this paper, we refer to a movie as an item and a user is a person who rates those movies. A known realization of the content-based approach is the Music Genome Project (Joyce, 2006) which is used for the internet radio service Pandora.

Another approach to finding the user's preferred items from an abundance of choices is by using the opinions of other people, which is known as the Collaborative Filtering approach. Collaborative Filtering (CF), also known as social filtering, recommends an item based on the past behavior of the users.

While the concept of sharing an opinion with others has been around mankind for centuries through their physical interaction, the term "Collaborative Filtering" was first coined by the developers of the first recommendation system – Tapestry (Goldberg et al., 1992).

Thanks to the Netflix, who offered a 1-million-dollar prize competition (Bennett and Lanning, 2007a) (Netflix, 2006). This helped to grab the interest and attention of many researchers to perform experiments and in-depth analysis of CF algorithms. Inspired by the competition, this work emphasizes using a collaborative filtering approach to recommendation system. A recommendation system is a subclass of an information filtering system which seeks to predict the unknown rating of a user-item pair, by discovering the correlation between users and items based on known user's explicit or implicit feedback. Recommendation systems are widely used and are very popular in both academia (Herlocker et al., 1999) (Konstan et al., 1997) and industry (Das et al., 2007) (Linden et al., 2003) research. Amazon (Linden et al., 2003) reported that in the year 2002, over 20% of Amazon's sales resulted from the personalized recommendation system. Currently, the number has soared to 30% and Amazon relies heavily on the recommendation system for its commercial success. While on the other side, 75% of videos played on Netflix are through the suggestions generated from the recommendation system, and YouTube is at 80%. (Breese et al., 1998) further elaborates and categorizes CF algorithms into two general classes namely, memory-based approach and model-based approach. The memory-based approach requires all the user, item and rating data to be stored on the memory for computation. Whereas, the model-based approach generates a summarization of user-item rating patterns periodically and saves them as a model. The model is then optimized using a machine learning algorithm.

The pure memory-based approach to CF is the neighborhood methods that are centered on computing the relationships between items or users. Further, the neighborhood methods are divided into the user-based and item-based approach. The user-based approach evaluates the preference of a user to an item based on the ratings provided to the item by the neighboring users. But, the item-based approach evaluates the preference of an item by a user based on the ratings of similar items by the same user. Neighborhood methods are further described and explored in chapter 3.

Latent factor models are the model-based approach to CF, which works by transforming both items and users to the same latent space, thus making them directly comparable. The latent factor tries to explain ratings by characterizing both items and users on factors automatically inferred from user feedback. Most of the successful realization of the latent factor model is based on matrix factorization (Koren et al., 2009).

With the experience of using the different algorithms in the Netflix challenge, we identified that there is no perfect model to generate an accurate rating (Bell and Koren, 2007b). The neighborhood methods are most effective in capturing the localized relationship between items and alternatively users. They rely on a few significant neighborhood relations, often ignoring many ratings by a user. Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items, but they are very weak in detecting associations between a small set of closely related items. Based on the experience of Netflix competition, there is no perfect model to be found. However, the best results came from combining the predictions of both models that complemented each other.

In this work, to acquire an accurate user-item rating, we define and use an abstract model which helps to incorporate both neighborhood methods and latent factor model together. The abstract model is called the factorization model which processes both algorithms simultaneously to generate an accurate rating. We start chapter 3 by defining the latent factor model, i.e matrix factorization model and then shift towards the neighborhood method. Finally, we define a factorization model that processes these algorithms in parallel. Chapter 3 also explains about the optimizing machine learning algorithm. The algorithm generates learning rules for each parameter of the factorization model and optimizes them for better accuracy.

The term to define the proliferation of content into the internet is known as ”Big Data”. The three perspectives of big data are volume, velocity, and variety. Volume refers to the amount of data being produced. There are 2.5 quintillion (10^{18}) bytes of data created each day at our current pace. Velocity is the speed of data being generated and processed with minimum error. Variety defines the type and nature of the data being produced. This exponential increase in data is the biggest challenge while developing the recommendation system. In order to handle large user and item data in the recommendation system, we use the MapReduce framework. The MapReduce framework which is developed by Google provides a parallel processing model. With MapReduce, the data is divided into blocks and distributed across nodes to process them in parallel. The results are then gathered together and combined for the final result. Chapter 4 is associated with the implementation of the MapReduce framework to factorization model using mrjob to process large user-item data. Chapter 5 shows the experiment of deploying the mrjob program into the Amazon Elastic MapReduce

(Amazon EMR). Amazon EMR is used to process big data across the Hadoop cluster of virtual servers on Amazon Elastic Computer Cloud (EC2) and Amazon Simple Storage Service (S3). The elastic in Amazon EMR's represent the dynamic resizing capacity, which allows setting the number of resources (EC2 instance) to use as per our demand. The experiments are performed on 100K, 1 million (1M), 10 million (10M)and 25 million (25M) MovieLens (GroupLensResearch, 2020) dataset using one, five and ten EC2 instances. This thesis is also presented in the 3 Minute Thesis Competition (Giri, 2020a) which was held on February 11, 2020 at the University of Central Missouri.

CHAPTER 2
LITERATURE REVIEW

Collaborative Filtering has seen a massive surge of interest in the past decade from both academic and industry due to the most popular 1-Million-Dollar Netflix Prize competition (Netflix, 2006). The surge started in October 2006 when an online DVD rental and video streaming service provider company Netflix announced an open competition for the best collaborative filtering algorithm that can predict the user rating to a movie based on the previous rating provided by the users. Anyone was able to take part in the competition and the first individual or team who could beat the then Netflix recommendation engine called CineMatch's accuracy by 10% would be the winner.

2.1 Neighborhood Methods

Many works have been done in the collaborative filtering field long before the Netflix competition was announced, dating back to the year 1992 with inventors of a manual document filtering recommendation system called Tapestry (Goldberg et al., 1992). The users of the Tapestry system had to manually give a set of queries to sort and filter the documents which they prefer to have. Not long after Tapestry, Paul and his associates at the University of Minnesota started their GroupLens project to examine automated collaborative filtering system, and eventually created Usenet news client (Konstan et al., 1997). Ringo music recommender (Shardanand and Maes, 1995) and, Bellcore's Video Recommender (Hill et al., 1995) expanded upon the GroupLens algorithm based on using a neighborhood-based method. Each of these systems follow a process of gathering ratings from users, computing correlations between pairs of users to identify the user's similar neighbors, and combining the ratings of those similar neighbors for generating recommendations to the users. Based on their individual experiments, both Ringo and Bellcore's Video Recommender supported GroupLens's idea to use the Pearson correlation coefficient metric to measure the correlation between users. Ringo limited membership in the neighborhood by only selecting the neighbors whose correlation was greater than a fixed threshold, Bellcore's Video Recommender calculated the correlation using the random sample of neighbors and the original GroupLens system used all available correlated neighbors to compute the final prediction. It is useful, both for accuracy and performance, to select a subset of users also known as, the neighbors to use in computing a prediction instead of the entire user database (Hill et al., 1995).

A consolidated study of using different similarity metrics and experiments by using different neighbor values for the user-based approach is detailed in (Herlocker et al., 1999). As the user-based algorithm for recommendation systems became more popular, their widespread use revealed some challenges. The two main challenges faced by the user-based approach were: sparsity and scalability. The user-based approach revealed poor recommendation accuracy when presented with large item data. In a commercial recommendation system, even an active user may have purchased well under 1% of the items and this generates the nearest neighbors of users very low and sometimes even revel no neighbors at all. Acknowledging the weakness of the user-based approach, (Sarwar et al., 2001) and (Linden et al., 2003) introduced an alternative approach named item-based methods. Instead of computing neighbors based on users, the item-based approach computes neighbors based on items and presents the items to users based on his/her purchase history. Systems implementing the item-based approach achieved a better prediction accuracy and was much faster. Experiments on (Bell and Koren, 2007a) and (Sarwar et al., 2001) provide strong evidence that the item-based approach delivers a more accurate prediction than a user-based approach. As such, we use item-based approach as one of the models in this work.

2.2 Latent Factor Models

Breese et al. (1998) divides collaborative filtering algorithms into two classes: memory-based algorithms and model-based approaches. The neighborhood-based approach to collaborative filtering is the memory-based approach whereas the model-based approach is known as the latent factor model. Latent factor models try to explain the ratings by characterizing the users and items in n number of factors inferred from the rating patterns. Some examples of latent factor models include (Hofmann, 2004), (Salakhutdinov et al., 2007) and (Blei et al., 2003). Matrix factorization techniques are a class of widely successful latent factor models that attempt to capture the low-rank approximation of the user-item matrix. The factorization technique takes in a utility user-item rating matrix and decomposes them into two low dimensional matrices in such a way that the product of those matrices generates the original utility matrix. The values of low dimensional matrices are called user and item latent factors and the dimensional of matrices determine the number of factors. In this work, we will focus on using the UV Decomposition method to find the low dimensional matrices.

There have been works on other matrix factorization domains such as Singular Value Decomposition (Deerwester et al., 1990), (Hofmann, 2004), (Billsus et al., 1998), (Goldberg et al., 2001), (Polat and Du, 2005), (Vozalis and Margaritis, 2007), (Gong et al., 2009) and Principal Component Analysis (Kim and Yum, 2005). Most of the collaborative filtering problems create a sparse matrix, and applying a matrix factorization technique to those matrix raises a problem of inaccurate prediction. Addressing only a few known ratings is highly prone to overfitting of the model. Works like (Kim and Yum, 2005) and, (Sarwar et al., 2000) depended on imputation to fill the missing ratings of sparse utility matrix to make the matrix dense. However, such activity increases the amount of data and the data could be considerably distorted due to the inaccurate imputation. Recent works like (Bell et al., 2007a), (Canny, 2002), (Funk, 2006), (Paterek, 2007), (Salakhutdinov et al., 2007), and (Takács et al., 2007) suggested modeling directly only on the observed rating while avoiding overfitting by using a regularized model. A gradient descent algorithm was then used to solve the problem (Bennett et al., 2007). (Bell et al., 2007a) acknowledged the other neighborhood issues and then provided an approach to overcome those shortcomings. However, this approach lead to a slow algorithm due to extensive training phase. (Salakhutdinov et al., 2007) also required an extensive training phase to generate an accurate rating.

2.3 Latent Factor Meets Neighborhood

An article by Bell and Koren (2007b) outlines an extensive summarization of the ideas of the team who won the first Netflix Prize. The article highlighted that the best results came from combining predictions of both neighborhood and latent factor models because they address the structure of data using a different strategy. This analysis led to more works such as (Bell and Koren, 2007c), (Bell et al., 2007a), and (Koren, 2008). (Bell et al., 2007b), and (Bell and Koren, 2007c) attempted to bring the neighborhood and latent factor models into one model for better accuracy. However, they suggested post-processing the factorization results, rather than a unified model where neighborhood and factorization models are considered symmetrically.

In this thesis, we focus on building a unified model which helps to process both neighborhood and latent factor model symmetrically. The model architecture is based on using the features derived from the data which even helps us to integrate other variants that effects the rating of an item.

2.4 Handling Large Scale Data

With the development of Web 2.0 (International and Media, 2004), the internet has seen a proliferation of information daily. One of the major challenges the recommendation system has to deal with is the large user and item data. Experiments presented by Chen et al. (2011) has adopted the technique to buffer the data into the memory so that the program does not need high memory to process the dataset. Another efficient way to handle large data is by using the MapReduce paradigm. Developed by Google (Dean and Ghemawat, 2008), MapReduce has been used in many domains to parallelly process the data by spreading them into a large pool of machines. (Das et al., 2007) have used MapReduce to implement the Probabilistic Latent Semantic Indexing (PLSI) algorithm to increase the speed and scalability. (Papadimitriou and Sun, 2008), the distributed co-clustering algorithm which parallelly searches for similar row and columns of paired data was implemented using MapReduce. A famous Apache project, called Mahout (Owen and Owen, 2012) dedicated to the design and implementation of scalable machine learning libraries, whose core algorithms for clustering and classification are implemented using MapReduce. Identifying the advantages of the MapReduce paradigm, in this thesis, we use MapReduce to handle the processing of large user and item data. In Chapter 3, we study the factorization model.

CHAPTER 3
FACTORIZATION MODEL

In this chapter, we present an overview of matrix decomposition, which forms the foundation for the factorization model. We also describe the extension to matrix decomposition model by integrating the neighborhood information which then forms our unified factorization model. Previous works (Bell and Koren, 2007c), (Bell et al., 2007b) and (Bell and Koren, 2007b) have discovered the advantages on bringing neighborhood and matrix decomposition models together to complement the limitations of each others as they address various levels of structures in the data. In this chapter, we understand the necessity and advantages of both models, and define an unified factorization model where both approaches are considered simultaneously. We begin the chapter by elaborating our approach and notation for the factorization model.

3.1 Approach

The factorization model uses matrix factorization as a primary model to overcome the data sparsity problem and neighborhood model as integration for identifying the local relationship between items or users. These two models can be run in a joint manner to give a rating score of a user's interaction to an item with minimum prediction error. The model-based approach to building a recommendation system takes in parameters which governs how the prediction is produced by the model. We used a machine learning algorithm to optimize those parameters and produce an accurate results. We start by defining the notations used in this chapter, data normalization techniques, matrix factorization model and neighborhood methods for collaborative filtering.

3.1.1 Notation

Suppose that we have M users and N items. We use u, v to denote users and i, j to denote items. $R \in \mathbb{R}^{M \times N}$ denotes the rating information, where r_{ui} is the rating given by user u to item i. The predicted rating given by user u to item i is denoted by $\hat{r}_{ui}$

3.2 Data Normalization

In the process of trying to predict the accurate $\hat{r}_{ui}$ using the models, we may come across extreme cases such as, large user and item effects - i.e, systemic tendencies for some users to give a higher rating than others and for some item to receive higher rating than others (Bell and Koren, 2007a) (Bell and Koren, 2007c). It is conventional to normalize our data by considering these effects.

The term we use to normalize those user and item effects are called *biases*, which are used to avoid predicting high for low rated items that happen to have a lot of neighbors with high average ratings and vice-versa. The *biases* neutralizes the prediction results that are systematically distorted due to faulty assumptions. A baseline model considering these user and item *biases* is defined as:

$$b_{ui} = \mu + b_u + b_i \tag{3.1}$$

here, we use b_{ui} to denote baseline predictor for user u item i rating r_{ui}. The parameters b_u and b_i, indicate the observed deviation of user u and item i respectively from the average μ. The value of average is the arithmetic mean of ratings $r \in \{1, 2, 3, 4, 5\}$, which a user assigns to an item. To better understand the baseline model of equation 3.1, we explain using a fabricated data. Let's suppose that we want a baseline estimate for the rating of the movie Frankenstein by user Bob. Considering average of all movies, μ is 3.2 stars. Furthermore, Frankenstein is a better than average movie and it tends to be rated 0.3 stars above average, which is consider as an item bias b_i. On the other hand, Bob is a critical user, who tends to rate 0.2 stars lower than average, which is the user bias b_u. Thus, the baseline estimate for Frankenstein's rating by Bob using equation 3.1 would be 3.3 stars based on following calculations:

$$b_{ui} = \mu + b_u + b_i$$

$$b_{ui} = 3.2 - 0.2 + 0.3$$

$$b_{ui} = 3.3$$

here, the baseline estimate b_{ui} is calculated based on three components, namely: global average μ , user bias b_u and item bias b_i. Considering these *biases* into the model has been proven to increase the accuracy based on experiments conducted on the paper (Bennett and Lanning, 2007b).

3.3 Matrix Factorization Model

Matrix factorization model is one of the state of the art models for large scale collaborative filtering. It works by characterizing users and items by latent features that are extracted from user-item rating matrix. The latent features are the numerical representation of raw data which describes the relationships between different aspects of data. Those latent features are not directly observed, but are inferred from other variables that are present in the data itself. Let's consider a latent feature space of two dimension, where we want to visu-

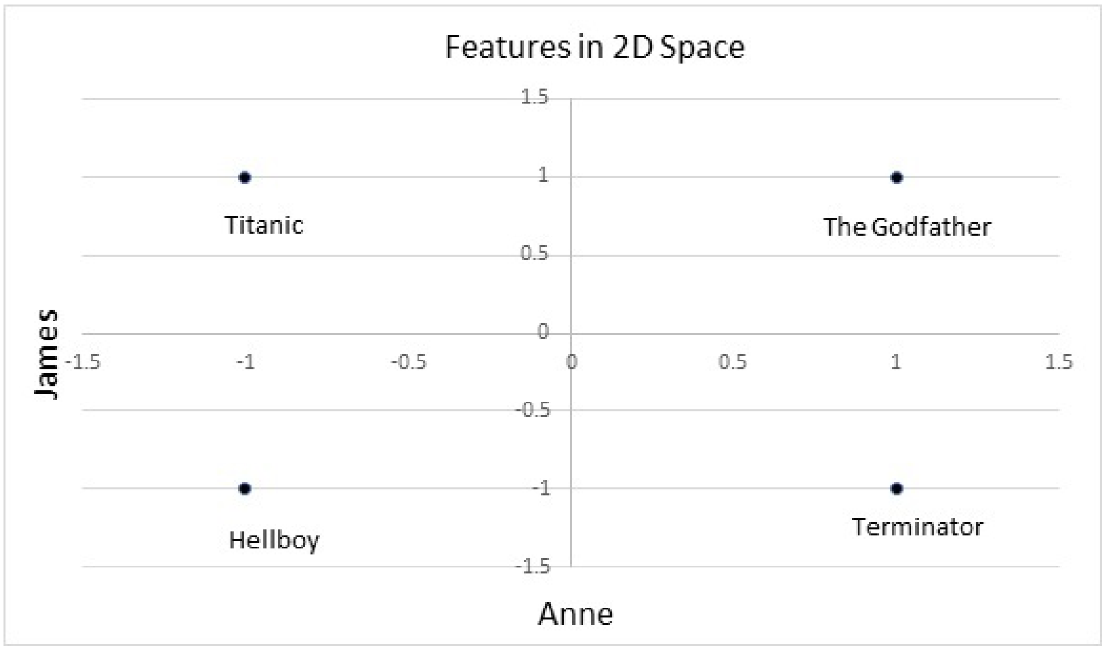


Figure 3.1: Features of user James and Anne in feature space.

alize a fabricated data. A fabricated data in this case are of two users James and Anne. James likes the movies such as "The Godfather", "Terminator" and hates movies such as "Hellboy", "Titanic" but in other hand, Anne likes movies such as "Titanic", "The Godfather" and hates the movies such as "Hellboy" , "Terminator". The feature space of both Anne and James are shown in Figure 3.1. Each movie is a point in the space of users. The dimension is the minimum number of coordinates required to specify the movies for user James and Anne, here are two [1,-1]. In matrix factorization model, the user-item rating matrix R is decomposed into two lower d dimension matrices which helps to capture the user-item factors in large sparse data. A typical collaborative filtering problem have very sparse data, meaning most of the ratings are unknown. Let's suppose the below table 3.1 where a user rates movie based on their degree of preference on a scale of 1-5, 5 being the highest. The missing rating for the movie is left

	Matrix	Titanic	Die Hard	Forrest Gump	Wall-E
Alice	5	1		2	2
Bob	1	5	2	5	5
Charlie	2		3	5	4
Diane	4	3	5	3	

Table 3.1: A fabricated data which consists of four users for five movies

blank in the Table 3.1 and we represent the Table 3.1 into matrix format as below:

$$R = \begin{bmatrix} 5 & 1 & & 2 & 2 \\ 1 & 5 & 2 & 5 & 5 \\ 2 & & 3 & 5 & 4 \\ 4 & 3 & 5 & 3 & \end{bmatrix}$$

In such sparse matrix, we do not have enough data to estimate the interactions between variables directly and independently. One way to estimate those bank ratings in the matrix R, is to assume the matrix R is actually the product of two long, thin matrices U and V. The process of finding these two thin matrices is called matrix factorization, and one of the approaches to discovering such two matrices is called UV-decomposition.

3.3.0.1 UV Decomposition

Most of the collaborative filtering datasets are highly sparse, for example: there are 99% missing ratings on MovieLens 25 million dataset, which means that most users respond to a very small number of movies. Maybe they like certain actor or actress over others, or they follow only certain directors, or they like certain genres - a general assumption would be that man likes more adventure or action movies where as women lean towards romantic genres. If we start with matrix R, with n rows and m columns (i.e, there are n users and m items), then we might be able to find a matrix U with n rows and d columns and a matrix V with d rows and m columns, such that the product of U and V closely approximates R. This process of breaking down the matrix into two thin matrices is called UV-decomposition of matrix R. A illustration of the process is described as follows:

$$\mathrm{R} \quad = \quad \mathrm{U} \quad \times \quad \mathrm{V}$$

$$\begin{bmatrix} 5 & 1 & & 2 & 2 \\ 1 & 5 & 2 & 5 & 5 \\ 2 & & 3 & 5 & 4 \\ 4 & 3 & 5 & 3 & \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \end{bmatrix} \times \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{bmatrix}$$

here, we have 4-by-5 rating matrix R with some missing values left as blank. We decompose matrix R into 4-by-2 and 2-by-5 matrices, U (user) and V (item), respectively. The matrix U has small number of columns and matrix V has small number of rows, so each are significantly smaller than matrix R, but they together represent most of the information presented in matrix R.

3.3.0.2 Initialization and Formulation

In order to identify the values of thin matrices U and V, which are used to approximate the matrix R, we start by giving each element of U and V the same value. We can use average value or if our data is normalized, we use zero. In this part, we initialize the values of U and V as random values and repeatedly adjust out values using an optimizing algorithm, which is discussed in the next section. For illustration using an example, we shall fill both thin matrices with random values and name them P and Q, where $Q \in \mathbb{R}^{item \times dimensions}$ and $P \in \mathbb{R}^{user \times dimensions}$. So essentially, we are trying to predict the rating $\hat{r}_{ui} \in \mathbb{R}^{users \times items}$ by user u to item i.

$$\begin{array}{ccccc} \mathrm{R} & = & \mathrm{P} & \times & \mathrm{Q} \end{array}$$

$$\begin{bmatrix} 5 & 1 & & 2 & 2 \\ 1 & 5 & 2 & 5 & 5 \\ 2 & ? & 3 & 5 & 4 \\ 4 & 3 & 5 & 3 & \end{bmatrix} = \begin{bmatrix} .1 & -.4 \\ -.5 & .6 \\ -2.4 & 1.1 \\ 1.1 & 2.1 \end{bmatrix} \times \begin{bmatrix} 1.1 & -.7 & .3 & .5 & .8 \\ -.8 & .8 & .5 & 1.4 & .3 \end{bmatrix}$$

Here, the values on matrix P and Q are called the factors of user and item respectively. These are the hidden features of both users and items inferred from the data and are called *latent factors*. Let's estimate the missing rating of user u to item i by calculating the dot product of user latent features (p_u) and item latent features (q_i) as:

$$\hat{r}_{ui} = p_u \cdot q_i^T \qquad (3.2)$$

where, p_u is the row u of matrix P and similarly, q_i is the column i of matrix Q. Using the equation 3.2, we calculate the dot product as:

$$\hat{r}_{ui} = p_u \cdot q_i^T$$

$$\hat{r}_{32} = p_3 \cdot q_2^T$$

$$\hat{r}_{32} = (-.7) \times (-2.4) + (1.1) \times (.8)$$

$$\hat{r}_{32} = 2.56$$

Hence, the missing rating value in matrix M, $\hat{r}_{32}$ is 2.56 - based on the latent factors inferred from the data. We call this method of identifying the rating using latent factors as matrix factorization technique. Now, combining the baseline estimate model from equation 3.1 and matrix factorization model from equation 3.2 we get the following model:

$$\hat{r}_{ui} = \mu + p_u \cdot q_i^T + b_u + b_i \tag{3.3}$$

3.4 Neighborhood Information

Neighborhood is the pre-dominant memory based approach widely used in both academia (Sarwar et al., 2001) (Herlocker et al., 1999) and industry (Linden et al., 2003) research area. These models are significantly popular because they are very effective in detecting the localized relationships between items and/or users. For a new item whose rating we want to predict, we scan through all the training items and choose several items that are most similar to the new item. Then, we predict the new item's rating value, based on the nearest neighbors' (known) ratings. The foundation for any neighborhood model is the similarity measure (s_{ij}) between two items and/or users, which measures correlation between them. A traditional method of computing neighborhood information using the user-based approach identifies pairs of items that tend to be rated similarly in order to predict ratings for an unrated items based on ratings of similar users. Further analyzing the user-based approach, experiments performed by (Sarwar et al., 2001) and (Linden et al., 2003) suggested the item-based approach provided better performance and accurate rating prediction than user-based approach. Algorithms centered around item-item similarity compute the user preference to an item based on their own ratings to similar items. The similarity between item i and item j is measured as the tendency of users to rate items i and j similarly. The common similarity measures are cosine, the adjusted cosine or the Pearson Correlation Coefficient.

In general, nearest neighborhood approach are often referred to by the shorthand k-NN, here k refers to the number of neighbors used, such as 3-NN and NN refers to nearest neighbors. The k value is the trade-off between accuracy and computation cost; meaning, as we increase k to the maximum possible (so that $k = n$) the entire dataset would be used for every prediction which then uses all the available memory for the computation. Hence, k is a free parameter which the user has freedom to choose.

3.5 Model Formulation

We start this section by initializing the $\hat{r}_{ui}$ under neighborhood method while adjusting for user and item effects through baseline estimates from 3.1 as:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in N(i,u;k)} s_{ij}(r_{uj} - b_{uj})}{\sum_{j \in N(i,u;k)} s_{ij}} \tag{3.4}$$

where, b_{ui} is the baseline estimate of user and item which normalizes the data for global effects. $N(i,u;k)$ is the set of k neighboring items rated by user u. The set $N(i,u;k)$ is computed by measuring the similarity between item i to other items rated by user u using Pearson Correlation Coefficient. The formula we use to calculate the pearson correlation coefficient (Statistics, 2006), denoted by ρ_{ij} is as follows:

$$\rho_{ij} = \frac{n\left(\sum xy\right) - \left(\sum x\right)\left(\sum y\right)}{\sqrt{\left[n\sum x^2 - \left(\sum x\right)^2\right]\left[n\sum y^2 - \left(\sum y\right)^2\right]}} \tag{3.5}$$

We precompute the similarity between all the items and pick the top k similar items for a user from the $N(i;u)$ set. The similarity values are measured on the scale of +1 to -1 for two items. The values $+1 \geq \rho_{ij} > 0$ indicates that there is a positive correlations, where if one value increases, then the other value increases too. The values between 0 > ρ_{ij} $\geq$ -1 indicates that there is a negative correlations, where is one value increases, the other value decreases. A special case where ρ_{ij} == 0 means, there is no correlation between two items. Further analyzing, a recent work (Bell and Koren, 2007c), raised number of concerns about the pure neighborhood model defined in equation 3.4 as:

- The similarity metric s_{ij} which directly defines the interpolation weights, is arbitrary. Various algorithms use different similarity measures to quantify the elusive notion of item similarity. It can be based on cosine , the adjusted cosine or the Pearson Correlation Coefficient.
- Also, the model does not account for interaction among neighbors. Each similarity between item i and a neighbor j $\in$ $N(i,u;k)$ is computed independently of the content of $N(i,u;k)$ and the other similarities: s_{ia} for $a \in N(i,u;k)$ - j. Suppose, that our items are movies that are highly correlated with each other (example: sequels such as "The Godfather I-III"). An algorithm that ignores the similarity of three movies when determining their interpolation weights might triple counting the information provided by the group.

To overcome the above concerns, Koren (2008) suggested an improved model which incorporates those difficulties. Given a set of neighbors $N(i,u;k)$ we need to compute interpolation weights $\theta^u_{ij} \,|\, j \in N(i,u;k)$ that enables the best prediction as follows:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in N(i,u;k)} \theta^u_{ij}(r_{uj} - b_{uj}) \tag{3.6}$$

The equation 3.6 calculates the similarity between item i and item j of set $N(i, u; k)$ which then helps to ease the second concern. We also note that, the above two models are centered around $user - specific$ interpolation weights - θ_{ij}^{u} in equation 3.6 and $\frac{s_{ij}}{\sum_{j \in N(i,u;k)} s_{ij}}$ in equation 3.4, relating item i to the items in a user-specific neighborhood $N(i, u; k)$. Since the similarity metrics used to compute the weights of above models are arbitrary, we modify the model to focus on global weights - independent of a specific users. We identify this global weight as w_{ij} and replace the $user - centered$ θ_{ij}^{u} of equation 3.6 as:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in N(i,u;k)} w_{ij}(r_{uj} - b_{uj}) \tag{3.7}$$

The weights until equation 3.6 represent an interpolation coefficient relating unknown ratings to existing ones, we now represent weights w_{ij} as an offsets to baseline estimates. With this assumption, we can view the residuals, $r_{ui} - b_{uj}$ as the coefficients multiplying those offsets. Hence, for two related items i and j, we expect w_{ij} to be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$) is high, we would like to increase our estimate for u's rating to i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline estimate. Likewise, our estimate will not deviate much from the baseline by an item j, that u rated just as expected (r_{uj}) is around zero, or by an item j that is not known to be predictive on i (w_{ij}) is close to zero. The parameter w_{ij} is learnt together along with the matrix factorization model parameters as suggested by (Bell and Koren, 2007c). Finally, we conclude with a more accurate model (Zheng et al., 2012) as:

$$\hat{r}_{ui} = b_{ui} + |N(i,u;k)|^{-1/2} \sum_{j \in N(i,u;k)} w_{ij}(r_{uj} - \bar{r}_u) \tag{3.8}$$

where, $N(i,u;k)$ is a set of items user u rate - items are selected as k-nearest neighbor of the item i and have been rated by user u. The parameter k is predefined, and if we use k as n, then k is the number of items in our dataset, where all item to item relations are explored. $|N(i,u;k)|$ is the cardinality of a set - value is the number of elements in the set. The parameter w_{ij} is the similarity measure from item i to j. We treat w_{ij} as a free parameter which is learnt together in matrix factorization model by an optimizing algorithm. The parameter $\bar{r}_u$ is the user average rating precalculated and the parameter r_{uj} is the true rating of item in set $N(i,u;k)$; rating given by user to item j.

3.5.0.1 Implementation

For a better grasp of the neighborhood model 3.8, we calculate the $\hat{r}_{ui}$ of a user to an item by using the fabricated dataset of table 3.1. Let's predict the rating of movie *Titanic* given by the user *Charlie* in the fabricated data table 3.1. Here, we have u as *Charlie*, i is *Titanic* and k as n (all the items of the dataset are included as neighboring item). With reference to item-based approach, we here calculate the similarity between item *Titanic* to all other items available in our table using the Pearson's formula 3.5. The item-item similarity calculation for the items rated by user *Charlie* to item Titanic is shown in table 3.2. We then sort the similarities yield by table 3.2 into descending or-

Movies	Similarity with Titanic
Matrix	$\frac{(3\times22)-(9\times10)}{\sqrt{(3\times35-9^2)(3\times42-10^2)}} \approx -0.96077$
Die Hard	$\frac{(2\times25)-(8\times7)}{\sqrt{(2\times34-8^2)(2\times29-7^2)}} \approx -1$
Forrest Gump	$\frac{(3\times36)-(9\times10)}{\sqrt{(3\times35-9^2)(3\times38-10^2)}} \approx +0.981$
Wall-E	$\frac{(2\times27)-(6\times7)}{\sqrt{(2\times26-6^2)(2\times29-7^2)}} \approx +1$

Table 3.2: Similarity of item Titanic to other items rated by user Charlie

der: $\{+1, +0.981, -0.96077, -1\}$. As we have defined k as n, the value of k is the number of items in our dataset i.e. 5. For simplicity, we breakdown the neighborhood model 3.8 into pieces and assign the values as:

- $N(u,i;k) = \{Wall - E, ForrestGump, Matrix, DieHard\}$
- $|N(u,i;k)| = 4$
- $\bar{r}_u = \dfrac{2+3+5+4}{4} = 3.5$

We now plug in the values to the equation 3.8 and perform further calculations as:

$$\begin{aligned}\hat{r}_{ui} &= \frac{1}{\sqrt{4}} \times \Big[(1 \times (4 - 3.5)) + (0.981 \times (5 - 3.5)) + (-0.96077 \times (2 - 3.5)) \\ &\quad + (-1 \times (3 - 3.5)) \Big] \\ &= 1.9563275\end{aligned}$$

Hence, based on the above approach (neighborhood method), the rating user *Charlie* will give to the item *Titanic* is 1.9563275. Note that we have used s_{ij} instead of w_{ij} to calculate the neighborhood information, however, w_{ij} value will be learnt using optimization algorithm as first suggested by (Bell and Koren, 2007c), which we explain in the next section.

3.6 Integrated Model

To implement the models designed in the previous section, we encounter two major challenges: First, there are many variants of models we want to experiment with and each of them have their own advantages and drawbacks. Secondly, the user-item rating matrix R is very large. To solve these two challenges while building a recommendation system, we design an unified toolkit. The toolkit takes in user and item data, process them simultaneously using both matrix factorization and neighborhood methods, and in order to handle large user and item data, we use MapReduce framework. MapReduce framework is explained in next chapter, we here derive a unified model which uses both matrix factorization and neighborhood methods.

As discussed on earlier sections, there are many algorithms for predicting user-item ratings. We also highlighted that memory based algorithm i.e, neighborhood methods are the most efficient in detecting the localized relationship but fail when there are very few or none known neighbors. In other hand, model based algorithm i.e, matrix factorization model is efficient in capturing global information and may have better generalization capabilities. In the next section, we will be focused on integrating both of these state-of-art algorithms into a single model where both approaches are considered simultaneously to generate an accurate rating.

3.6.0.1 Linear Regression

The very basic collaborative filtering model used traditionally is the baseline estimate which we discussed in the section 3.2. An another approach for predicting the values of collaborative filtering problem would be neighborhood model which we described in the section 3.4. Let's start with the refined neighborhood model from equation 3.8:

$$\hat{r}_{ui} = \mu + b_u + b_i + |N(i,u;k)|^{-1/2} \sum_{j \in N(i,u;k)} s_{ij}(r_{uj} - \bar{r}_u) \tag{3.9}$$

The above model has three main components namely : global mean value μ , user and item effects $b_u + b_i$ and neighborhood model $|N(i,u;k)|^{-1/2} \sum_{j \in N(i,u;k)} s_{ij}(r_{uj} - \bar{r}_u)$. Implementing all three components individually and merging them together seems to be a tedious process. However, if we compare these three models, we notice that all three models are special case of linear regression problem as:

$$y = \sum_i w_i x_i \tag{3.10}$$

The ordered list of numeric features are known as a vector and a vector can be visualized as a point in a feature space. As illustrated in the Figure 3.1, we have a two dimensional vector [1,-1] representing James and Anne's preference of movies. Then, a feature vector is an d dimensional vector of numerical features that represent users and items. Suppose we have n users, m items and h total number of possible s_{ij} in equation 3.9. We can define the feature vector $x = [x_0, x_1, \ldots, x_{n+m+h}]$ for user item pair $< u, i >$ as follows:

$$x_k = \begin{cases} Indicator(u == k) & if \quad k < n \\ Indicator(i == k - n) & if \quad n \leq k < n + m \\ 0 & if \quad k \geq m + n, j \notin N(i, u; k), s_{ij} \text{ means } w_k \\ |N(i, u; k)|^{-1/2}(r_{uj} - \bar{b}_u) & if \quad k \geq m + n, j \notin N(i, u; k), s_{ij} \text{ means} w_k \end{cases}$$

The choice of pairs s_{ij} can be flexible and we can choose only possible neighbors instead of all the pairs. The corresponding layout of weight w will be:

$$w = \left[b_u(0), b_u(1), \ldots, b_u(n), b_i(m) \ldots s_{ij} \ldots\right] \tag{3.11}$$

Ultimately, the equation 3.9 can be reformed as:

$$\hat{r}_{ui} = \mu + b_u 1 + b_i 1 + \sum_{j \in N(i,u;k)} s_{ij} \left[|N(i, u; k)|^{-1/2}(r_{uj} - \bar{r}_u)\right] \tag{3.12}$$

where b_u, b_i, s_{ij} corresponds to weight of linear regression, and the coefficients on the right of the weight are the input features. Under this framework, we need to layout the parameters into a feature vector. In our case, we arrange first n features to b_u then b_i and s_{ij}, then transform the input data into the format of linear regression input. Finally, we use a linear solver to solve the problem (Chen et al., 2011).

3.6.0.2 Feature-Based Factorization Model

In this section, we discuss feature-based generalization for matrix factorization model. A basic matrix factorization model with baseline estimate is:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u \cdot q_i^T \tag{3.13}$$

This equation 3.13 is the combination of equations 3.1 and 3.2 where, p_u models the latent preference of user u and qi models the latent property of item i. Then the direct generalization of the model will be:

$$\hat{r}_{ui} = \mu + \sum_j w_j x_j + b_u + b_i + p_u \cdot q_i^T \tag{3.14}$$

The equation 3.14 adds the linear regression term 3.10 to the basic matrix factorization model shown in the equation 3.13. This allows us to add more bias information, such as neighborhood information. A more direct way to make our model more flexible to include other factors is to use features in factors as well. Instead of implementing each variant of matrix factorization models, we define a toolkit to solve the following abstract model:

$$y(\alpha,\beta,\gamma) = \mu + \left(\sum_j b_j^{(g)}\gamma_j + \sum_j b_j^{(u)}\alpha_j + \sum_j b_j^{(i)}\beta_j\right) + \left(\sum_j p_j\alpha_j\right)^T \cdot \left(\sum_j q_j\beta_j\right) \tag{3.15}$$

here, the model consists of three kinds of features $\langle\alpha,\beta,\gamma\rangle$ where, α denotes user feature, β denotes item feature and γ is global feature, b^g is global bias, b^u is user bias and b^i is item bias. Using the model 3.15 as a abstract model, we can now fit most of the models that we described in the previous sections. Basic matrix factorization model is a special case of the equation 3.15. For predicting user item pair $< u, i >$, we define:

$$\gamma_h = \varnothing, \qquad \alpha_h = \begin{cases} 1 & if \quad h = u \\ 0 & if \quad h \neq u \end{cases}, \qquad \beta_h = \begin{cases} 1 & if \quad h = i \\ 0 & if \quad h \neq i \end{cases}$$

The global feature γ allows us to add more bias information which directly affects users' preferences, such as neighborhood information. To include the neighborhood information into the factorization model, we redefine γ as below:

$$\gamma_h = \begin{cases} \dfrac{r_{uj} - \bar{r}_u}{\sqrt{|N(i,u;k)|}} & if \quad h = index(i,j), \quad j \in N(i,u;k) \\ 0 & if \qquad\qquad otherwise \end{cases}$$

here, index maps the possible pairs in k-nearest neighborhood set to consecutive integers. Using the fabricated dataset from Table 3.1, we calculate the γ_h taking h as index (Titanic, Matrix) for user Charlie. Here, we have i as *Titanic*, j as *Matrix* and similarly, r_{uj} as 2. For the user *Charlie*, we have user average ratings ($\bar{r}_u$) as (2 + 3 + 5 + 4)/4 = 3.5. We then compute the value of γ as:

$$\begin{aligned}\gamma_h &= \frac{r_{uj} - \bar{r}_u}{\sqrt{|N(i,u;k)|}} \\ &= \frac{2 - 3.5}{\sqrt{4}} \\ &= -0.75\end{aligned}$$

Likewise, using the same approach, we precompute the index (Titanic, Die Hard) as -0.25 and index (Titanic, Forrest Gump) as 0.75. Based on the user's choice of k, we select the index for the user Charlie. The model parameter set is defined as $\Theta = \{b^g, b^u, b^i, \mathrm{p}, \mathrm{q}\}$ where, $\mathrm{p}_j \in \mathbb{R}^d$ and $\mathrm{q}_j \in \mathbb{R}^d$ are d dimensional latent factors associated with each features. Similarly, $\mathrm{p}_j \cdot \alpha_j$ is the dot product between user latent factor and user feature, $\mathrm{q}_j \cdot \beta_j$ is the dot product between item latent factor and item feature. Intuitively, we use a linear model to construct user and item latent factors from features. The parameters of set Θ are trained efficiently by minimizing the loss function using optimizing algorithm which is discussed in next section.

3.7 Learning Algorithm

The feature-based factorization model represented in equation 3.15, is very flexible as it allows us to have many parameters. The parameters determines the complexity and accuracy of the model, which is used for the prediction of the rating ($\hat{r}_{ui}$). We measure the performance of factorization model using root mean squared error (RMSE). On a test dataset T, which consists of user-item rating, the RMSE is computed using the following formula:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T}(r_{ui} - \hat{r}_{ui})^2}{|T|}} \tag{3.16}$$

here, lower RMSE value corresponds to higher prediction accuracy. To gain the less RMSE value, we tune our model's parameters present in set Θ to reduce the cost function. We start the section with cost function and then define a learning algorithm.

3.7.0.1 Cost Function

The generated factorization model helps to incorporate neighborhood information into matrix factorization for prediction accuracy. Here, we define a cost function which measures how well the predicted rating by the model for an user-item dataset matches the original rating. The below equation is the cost function for the model, and it includes a regularization term at the end to prevent the model from overfitting. A overfitting occurs when the model fits training dataset perfectly and results a large error on testing dataset. Our goal is to measure the disparity of predicted rating with the original rating using cost function and minimize the cost function to improve the accuracy of the model. The cost function for model is as:

$$J(\Theta) = \frac{1}{2 \times n} \times (y - \hat{y})^2 + regularization \tag{3.17}$$

where, n is the number of training samples, y is the true rating recorded in the dataset and $\hat{y}$ is the rating generated by the factorization model given in the equation 3.15. We use linear function, $f(.)$ which takes y as input and provides $\hat{y}$ as an output. The final version of the models using linear function will be as :

$$\hat{y} = f(y) \tag{3.18}$$

To prevent the model from overfitting, we need to regularize each parameters of set Θ. We then form a new equation which adds these parameters to the original factorization model of equation 3.15:

$$y = \mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j \right) + \left(\sum_j p_j \alpha_j \right)^T \cdot \left(\sum_j q_j \beta_j \right)$$
$$+ \lambda \left(||b_j^{(g)}||^2 + ||b_j^{(u)}||^2 + ||b_j^{(i)}||^2 + ||p_j||^2 + ||q_j||^2 \right) \quad (3.19)$$

here, the constant λ is known as regularization term. All the parameters of the set Θ is multiplied by λ, to prevent the model from overfitting.

3.7.0.2 Model Learning

From the above section, we now have a model which predicts a rating. In this section, we use an optimization algorithm which runs iteratively to minimize the cost function mentioned in equation 3.17. The algorithm is called gradient descent algorithm. The algorithm finds the derivative of the cost function, which then helps us to decide if we need to increase or decrease the values of the parameters to get the minimum cost function. Then, the minimum cost function yields the accurate rating. Using this approach, in each step, the full gradient over all training data is computed and is used to update the parameters. However, using full gradient in each training iteration has a slow convergence as it needs to run optimization for whole training ratings in each iteration. As such, we use another flavor of gradient descent known as stochastic gradient descent. Now, this algorithm iterates over one rating at a time and updates the model parameters. We start the Algorithm 1 by initializing the parameters of set Θ

Algorithm 1 Stochastic Gradient Descent

```
initialize the parameters of set $\Theta$;
repeat:
$\Theta_j = \Theta_j - \eta \frac{\partial J(\Theta)}{\partial \Theta_j}$;
until converge;
```

as zeros and keep updating those parameters to reduce the cost function J(Θ). At each iteration, we follow the gradient direction with step size η. We repeat the process until the model converges or the training round is over. $\eta \in \mathbb{R}$ is the size of steps we take to in order to reach the minimum cost function. We can reach minimum cost function by taking large step size but might run into the chances of overshooting the minimum. On the other hand, smaller steps takes alot of times to reach the minimum cost function.

3.8 Update Rules

Stochastic gradient descent algorithm works by finding the derivative of the parameters. In this section, we calculate the partial derivative of the parameters present in set Θ. The derivative then forms the update rule for those parameters. Here, $\hat{e} = y$ - $\hat{y}$, the difference between original rate and predicted rate which is used in the next phase for generating update rules.

3.8.0.1 Generating Update Rule for User Factor

$$
\begin{aligned}
p_i &= p_i - \left(\eta \times \left(\frac{\partial}{\partial p_i} \times \frac{1}{2} \times (y - \hat{y})^2 + \lambda \times p_i\right)\right) \\
&= p_i - \left(\eta \times \left(\frac{1}{2} \times 2 \times (y - \hat{y})^{2-1} \times \frac{\partial}{\partial p_i} \times (y - \hat{y}) + \lambda \times p_i\right)\right. \\
&= p_i - \left(\eta \times \left((y - \hat{y}) \times \frac{\partial}{\partial p_i} (y - \hat{y}) + \lambda \times p_i\right)\right) \\
&= p_i - \Biggl(\eta \times \hat{e} \times \frac{\partial}{\partial p_i} \Biggl(y - \Biggl(\mu + \Biggl(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\Biggr) \\
&\quad + \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right) + \lambda \times \eta \times p_i \\
&= p_i - \Biggl(\eta \times \hat{e} \times \Biggl(\frac{\partial}{\partial p_i} (y) - \frac{\partial}{\partial p_i} (\mu) - \frac{\partial}{\partial p_i} \Biggl(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\Biggr) \\
&\quad - \frac{\partial}{\partial p_i} \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right) + \lambda \times \eta \times p_i \\
&= p_i - \left(\eta \times \hat{e} \times \left(-\alpha_i \times \sum_j q_j \beta_j\right) + \lambda \times \eta \times p_i\right) \\
&= p_i + \left(\eta \times \hat{e} \times \left(\alpha_i \times \left(\sum_j q_j \beta_j\right)\right) - \lambda \times \eta \times p_i\right) \\
&= p_i + \eta \left(\hat{e} \alpha_i \left(\sum_j q_j \beta_j\right) - \lambda p_i\right)
\end{aligned}
$$

3.8.0.2 Generating Update Rule for Item Factor

$$
\begin{aligned}
q_i &= q_i - \left(\eta \times \left(\frac{\partial}{\partial q_i} \times \frac{1}{2} \times (y - \hat{y})^2 + \lambda \times q_i\right)\right) \\
&= q_i - \left(\eta \times \left(\frac{1}{2} \times 2 \times (y - \hat{y})^{2-1} \times \frac{\partial}{\partial q_i} \times (y - \hat{y})\right) + \lambda \times q_i\right) \\
&= q_i - \left(\eta \times \left((y - \hat{y}) \times \frac{\partial}{\partial q_i} (y - \hat{y}) + \lambda \times q_i\right)\right) \\
&= q_i - \left(\eta \times \hat{e} \times \frac{\partial}{\partial q_i} \left(y - \left(\mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\right)\right.\right.\right. \\
&\quad \left.\left.\left. + \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right)\right)\right) + \lambda \times \eta \times q_i\right) \\
&= q_i - \left(\eta \times \hat{e} \times \frac{\partial}{\partial q_i} (y) - \frac{\partial}{\partial q_i} (\mu) - \frac{\partial}{\partial q_i} \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\right)\right. \\
&\quad \left. - \frac{\partial}{\partial q_i} \left(\left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right)\right) + \lambda \times \eta \times q_i\right) \\
&= q_i - \left(\eta \times \hat{e} \times \left(-\beta_i \times \sum_j p_j \alpha_j\right) + \lambda \times \eta \times q_i\right) \\
&= q_i + \left(\eta \times \hat{e} \times \left(\beta_i \times \sum_j p_j \alpha_j\right) - \lambda \times \eta \times q_i\right) \\
&= q_i + \eta \left(\hat{e} \beta_i \left(\sum_j p_j \alpha_j\right) - \lambda q_i\right)
\end{aligned}
$$

3.8.0.3 Generating Update Rule for Global Bias

$$
\begin{aligned}
b_i^{(g)} &= b_i^{(g)} - \left(\eta \times \left(\frac{\partial}{\partial b_i^{(g)}} \times \frac{1}{2} \times (y - \hat{y})\right)^2 + \lambda \times b_i^{(g)}\right) \\
&= b_i^{(g)} - \left(\eta \times \frac{1}{2} \times 2 \times (y - \hat{y}) \times \frac{\partial}{\partial b_i^{(g)}} (y - \hat{y}) + \eta \times \lambda \times b_i^{(g)}\right) \\
&= b_i^{(g)} - \left(\eta \times \hat{e} \times \frac{\partial}{\partial b_i^{(g)}} \left(y - \left(\mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\right)\right.\right.\right. \\
&\quad + \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right) + \lambda \times \eta \times b_i^{(g)} \\
&= b_i^{(g)} - \left(\eta \times \hat{e} \times -\gamma_i + \lambda \times \eta \times b_i^{(g)}\right) \\
&= b_i^{(g)} + \left(\eta \times \hat{e} \times \gamma_i - \lambda \times \eta \times b_i^{(g)}\right) \\
&= b_i^{(g)} + \eta \left(\hat{e}\gamma_i - \lambda b_i^{(g)}\right)
\end{aligned}
$$

3.8.0.4 Generating Update Rule for User Bias

$$
\begin{aligned}
b_i^{(u)} &= b_i^{(u)} - \left(\eta \times \left(\frac{\partial}{\partial b_i^{(u)}} \times \frac{1}{2} \times (y - \hat{y})\right)^2 + \lambda \times b_i^{(u)}\right) \\
&= b_i^{(u)} - \left(\eta \times \frac{1}{2} \times 2 \times (y - \hat{y}) \times \frac{\partial}{\partial b_i^{(u)}} (y - \hat{y}) + \eta \times \lambda \times b_i^{(u)}\right) \\
&= b_i^{(u)} - \left(\eta \times \hat{e} \times \frac{\partial}{\partial b_i^{(u)}} \left(y - \left(\mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\right)\right.\right.\right. \\
&\quad \left.\left.\left. + \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right)\right)\right) + \lambda \times \eta \times b_i^{(u)}\right) \\
&= b_i^{(u)} - \left(\eta \times \hat{e} \times -\alpha_i + \lambda \times \eta \times b_i^{(u)}\right) \\
&= b_i^{(u)} + \left(\eta \times \hat{e} \times \alpha_i - \lambda \times \eta \times b_i^{(u)}\right) \\
&= b_i^{(u)} + \eta \left(\hat{e}\alpha_i - \lambda b_i^{(u)}\right)
\end{aligned}
$$

3.8.0.5 Generating Update Rule for Item Bias

$$\begin{aligned}
b_i^{(i)} &= b_i^{(i)} - \left(\eta \times \left(\frac{\partial}{\partial b_i^{(i)}} \times \frac{1}{2} \times (y - \hat{y})\right)^2 + \lambda \times b_i^{(i)}\right) \\
&= b_i^{(i)} - \left(\eta \times \frac{1}{2} \times 2 \times (y - \hat{y}) \times \frac{\partial}{\partial b_i^{(i)}} (y - \hat{y}) + \eta \times \lambda \times b_i^{(i)}\right) \\
&= b_i^{(i)} - \left(\eta \times \hat{e} \times \frac{\partial}{\partial b_i^{(i)}} \left(y - \left(\mu + \left(\sum_j b_j^{(g)} \gamma_j + \sum_j b_j^{(u)} \alpha_j + \sum_j b_j^{(i)} \beta_j\right)\right.\right.\right. \\
&\quad + \left(\sum_j p_j \alpha_j\right) \cdot \left(\sum_j q_j \beta_j\right) + \lambda \times \eta \times b_i^{(i)} \\
&= b_i^{(i)} - \left(\eta \times \hat{e} \times -\beta_i + \lambda \times \eta \times b_i^{(i)}\right) \\
&= b_i^{(i)} + \left(\eta \times \hat{e} \times \beta_i - \lambda \times \eta \times b_i^{(i)}\right) \\
&= b_i^{(i)} + \eta \left(\hat{e}\beta_i - \lambda b_i^{(i)}\right)
\end{aligned}$$

3.8.0.6 Generating Update Rule for Neighborhood Parameter

$$\begin{aligned}
w_{ij} &= w_{ij} - \left(\eta \times \left(\frac{\partial}{\partial w_{ij}} \times \frac{1}{2} \times (y-\hat{y})\right)^2 + \lambda \times w_{ij}\right) \\
&= w_{ij} - \left(\eta \times \frac{1}{2} \times 2 \times (y-\hat{y}) \times \frac{\partial}{\partial w_{ij}} (y-\hat{y}) + \lambda \times w_{ij}\right) \\
&= w_{ij} - \left(\eta \times (y-\hat{y}) \times \frac{\partial}{\partial w_{ij}} \left(y - |N(u,i;k)|^{-1/2} \sum_{j\in N(u,i;k)} w_{ij}\left(r_{uj} - \bar{r}_u\right) \times \lambda \times w_{ij}\right)\right) \\
&= w_{ij} - \left(\eta\left(\hat{e}\left(-|N(u,i;k)|^{-1/2} \sum_{j\in N(u,i;k)} \left(r_{uj} - \bar{r}_u\right) + \lambda \times w_{ij}\right)\right)\right) \\
&= w_{ij} + \eta\left(\hat{e} \times |N(u,i;k)|^{-1/2} \sum_{j\in N(u,i;k)} \left(r_{uj} - \bar{r}_u\right) - \lambda \times w_{ij}\right)
\end{aligned}$$

Hence, from the above derivative, update rule for the parameters of set Θ and neighborhood model's ***wij*** of equation 3.8 are:

$$w_{ij} = w_{ij} + \eta\left(\hat{e}|N_{(u,i;k)}|^{-1/2} \sum_{j\in N(u,i;k)} \left(r_{uj} - \bar{r}_u\right) - \lambda w_{ij}\right) \tag{3.20}$$

$$p_i = p_i + \eta\left(\hat{e}\alpha_i\left(\sum_j q_j\beta_j\right) - \lambda p_i\right) \tag{3.21}$$

$$q_i = q_i + \eta \left(\hat{e}\beta_i \left(\sum_j p_j \alpha_j \right) - \lambda q_i \right) \tag{3.22}$$

$$b_i^{(g)} = b_i^{(g)} + \eta \left(\hat{e}\gamma_i - \lambda b_i^{(g)} \right) \tag{3.23}$$

$$b_i^{(u)} = b_i^{(u)} + \eta \left(\hat{e}\alpha_i - \lambda b_i^{(u)} \right) \tag{3.24}$$

$$b_i^{(i)} = b_i^{(i)} + \eta \left(\hat{e}\beta_i - \lambda b_i^{(i)} \right) \tag{3.25}$$

We use the above generated update rule to train our model's parameters repetitively until the model converges. In the next chapter, we describe the MapReduce paradigm and use it to solve the factorization model.

CHAPTER 4
MAPREDUCE

MapReduce is a programming framework that processes a large volume of data in-parallel on clusters of commodity servers in a fault-tolerant manner. MapReduce was first developed by Google and is used for the generation of data for Google's production web search service, sorting, data mining, large scale machine learning, and many other systems (Dean and Ghemawat, 2008). MapReduce can be applied to significantly larger datasets than a single commodity server can handle - a large data center can use MapReduce to sort a petabyte of data in only a few hours (Cardosa et al., 2011). In this chapter, we will concentrate on implementing the MapReduce framework to overcome the problem of processing large user and item data in the factorization model.

4.1 Architecture

MapReduce abstraction is inspired by the *map* and *reduce* primitives originally present in Lisp and many other functional programming languages. MapReduce works by breaking the processing of complex computation on large user and item data into two phases: the *map* phase and *reduce* phase. Each phase

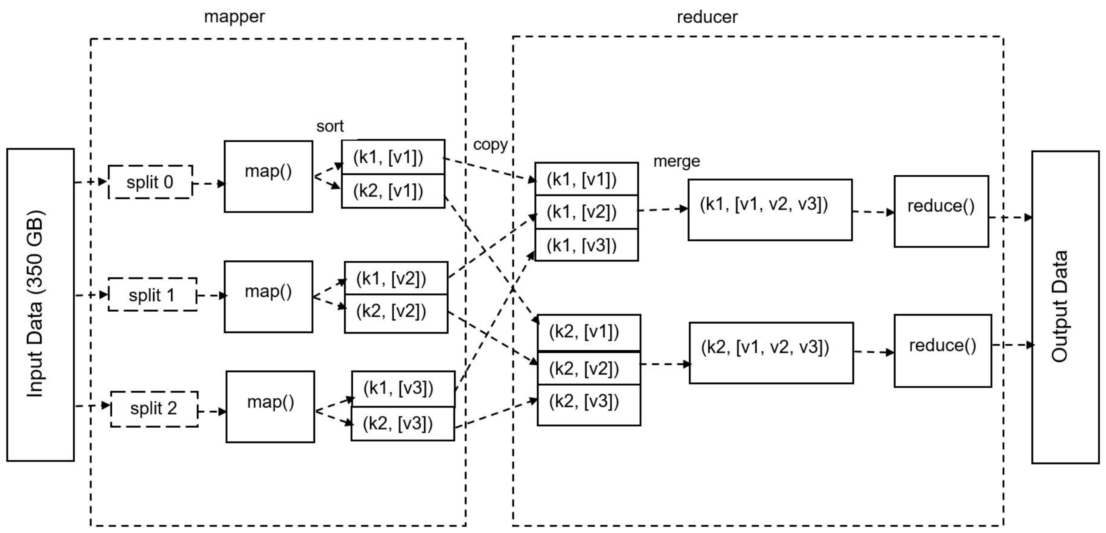


Figure 4.1: MapReduce Programming Model Architecture (White, 2009)

has key-value pairs as input and output. Within the *map* and *reduce* phase, a programmer specify a *map* function that processes a key-value pair to generate a set of intermediate key-value pairs, and a *reduce* function merges all intermediate values associated with the same intermediate key. The next section of the chapter explains the main component which forms MapReduce framework architecture. A pictorial representation of the MapReduce architecture is shown in Figure 4.1.

4.1.1 Mapper

The function of the mapper starts by reading the input file from standard input (stdin) and dividing the input into an appropriate block called splits.

The framework assigns one *map* task for each split, which runs the user-defined *map* function and writes their output to the local disk. The number of *map* task is equal to the number of the split created, i.e, the total number of blocks of the input files. For example: if we have 350 GB of data and have split size of 128 MB then, we will have 3 *maps*. *X* input files will generate *Y map* tasks to be run and each *map* task will generate as many output files as it needs. The output files will be grouped in a manner where all the key-value pairs for a given key end up in files targeted at a specific reduce task.

4.1.2 Reducer

Input to the single reducer is the output from all the *maps*. Each *reduce* function reduces a set of intermediate values that share a key to a smaller set of values. The number of reduce tasks is arbitrary and the larger number of reducers makes the reduce phase faster, since we get more parallelism. The dataflow for case of multiple reduce tasks is illustrated in Figure 4.1.

4.1.3 Combiner

Often times, the cluster of commodity machines has a limited bandwidth available, which can highly effect the processing time for the MapReduce jobs. This is because the output of each *map* task is present on the local machine's hard drive and they need to be transferred to the machine where the *reduce* task is running for further processing. The framework allows us to specify a *combiner* function, to be run on the *map* output while it is still in the memory to compress the map output. The combiner function results a more compact output, so there is less data to write on the disk and to transfer to the reducer. The output of *combiner* function forms the input to *reduce* function.

When *map* task outputs its key-value pairs, we initiate a *combiner* function which gathers the key-value pairs emitted from *map* function and stores them in an in-memory list instead of writing them on the hard disk. The number of the list created depends on the number of unique keys being generated from the *map* function. The *combiner* finally emits the processed key-value pairs to *reducer* function for further processing. The general form of map, combiner and reduce is as follows:

map : (k1,v1) $\rightarrow$ list(k2,v2)

combiner : (k2, list(v2)) $\rightarrow$ list(k2,v2)

reduce : (k2,list(v2)) $\rightarrow$ list(k3,v3)

The mapper is a generic type which allows it to work with any key or value types. The map input key and value types (k1 and v1) are different from the map output types (k2 and v2). The output types of combiner are the intermedi-

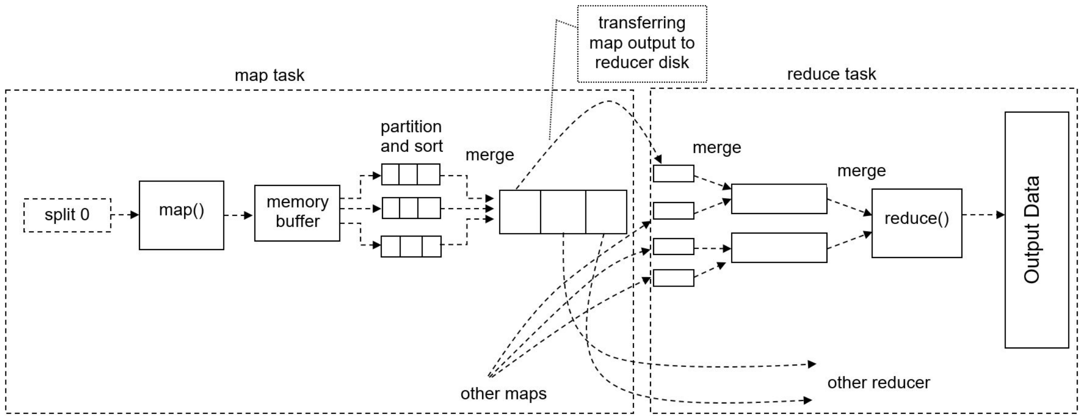


Figure 4.2: Shuffle and Sort Workflow Architecture (White, 2009)

ate key and value types (k2 and v2), which are sent as input to reduce function. The reduce input must have the same types as the map output and the reducer output types are different (k3 and v3).

4.1.4 Shuffle and Sort

The MapReduce framework makes certain that the input to every reduce task is sorted by key. The process where the framework implements sort and transfers the map key-value pair outputs to the reducers as an input is called shuffle. The shuffle and sort operation is the heart of MapReduce framework and the operations are performed on both map and reduce phase. The function of these operands on map and reduce phase is illustrated on the Figure 4.2 and each of their roles are summarized below:

4.1.4.1 Shuffle and Sort - A Map View

When map task starts producing the outputs, it is initially present in the memory buffer and before it gets written into the local disk, the framework divides the data into partitions corresponding to each reducers that they will be sent to. Within each partition, the framework performs an in-memory sort by keys, and if there is a *combine* function, it is run on the output of the sort. Once, all the map task has emitted its output, the pieces of partitions are merged sorted into a single partition file and written into the local disk of the mapper. The output file's partition are made available to the reducers over HTTP.

4.1.4.2 Shuffle and Sort - A Reduce View

At present, the map output file is held on the local disk of the machine that ran the map task. Now, the transfer of file occurs as the reduce task needs the map output for its particular partition from several map task across the cluster. The sort phase of reduce side sorts and merges the map outputs, maintaining their sort ordering. The reduce function is called for each key in the sorted output and the output of the reduce phase is written to the standard output (stdout). On the large cluster of commodity servers, we can parallelly run the *map* function on the servers where the data resides, rather than copying the data to the destination of the computer program. The output of *map* task is saved into the individual server and the *reducer* task can be run to merge the results on those servers in parallel. The *mapper* and *reducer* are both executable that reads an input line by line from the standard input (stdin) and writes output to the standard output (stdout). To leverage the functionality of MapReduce framework in factorization model, we write a python program using mrjob package. The details on the use of mrjob in the factorization model and its use cases are elaborated on next section.

4.2 mrjob

The software package mrjob is a MapReduce library created by Yelp that allows us to write MapReduce application programs using a pure python programming language in our local machine. Using the mrjob library in python, we write a multi-step MapReduce job where the output of one *reducer* acts as an input to another job's *mapper* or *reducer*. This multi-step functionality allows us to write a MapReduce program in a single class, instead of writing separate programs for the *mapper* and *reducer*. In the next section, we explain the MapReduce job using mrjob library in python programming language for the factorization model. The program can be tested locally without having Hadoop installed on the local machine and can be run on the cloud using Amazon Elastic MapReduce (EMR). The program is designed to be flexible and scalable while running on EMR, a user can define the number of *maps* to use, the type of Amazon Elastic Compute Cloud (EC2) instance to use as a *map* and the geographical region to deploy the EMR cluster. Once the program starts running, it automatically creates an EMR cluster based on the settings defined on the configuration file and terminates the EMR cluster automatically when the job is completed successfully or report errors.

4.2.1 Factorization Model using mrjob

One of the major challenges of any recommendation system is to manage large *user* and *item* data and process them quickly. According to 2018 data report (Lisa Richwine, 2018), the popular media-service provider and production company - Netflix have 137 million *users* and 45 thousand *items* in its recommendation system. With years of experience within the industry, Netflix further mentioned that a *user* tends to lose interest and deviate towards another site if they do not find any interesting content within 60 to 90 seconds(Netflix Technology Blog, 2016). To capture the *user's* attention within such a short time and to handle such a large *user* and *item* data, becomes a challenging task for any organization implementing recommendation system. In this section, we integrate mrjob into factorization model to process complex computation of large *user* and *item* data faster and efficiently.

We use the dataset published by GroupLens Research (GroupLensResearch, 2020), which has rating datasets from the decades. The dataset contains large volume of ratings provided by users to movies, making it a good candidate for analysis with MapReduce framework. The data are semi-structured and stored using line-oriented ASCII format, in which each line is a record. We write a mrjob program for factorization model where it takes MovieLens dataset as input and predicts the rating of a movie..

4.2.1.1 Architecture

A parallel implementation of the feature-based factorization model using mrjob is shown in Figure 4.3. A MapReduce job is a unit of work that a user wants to be performed, it consists of input data, the MapReduce program (mrjob program), and configuration file which contains the settings of the cluster employing the MapReduce job. The raw data is first converted into feature format data and then divided into a training set and testing set. Using 5 fold cross-validation, the training set and the testing set are divided into 80% and 20% of the original dataset respectively, where the training set helps to build the model, and the testing set validates the model. To evaluate our model, we

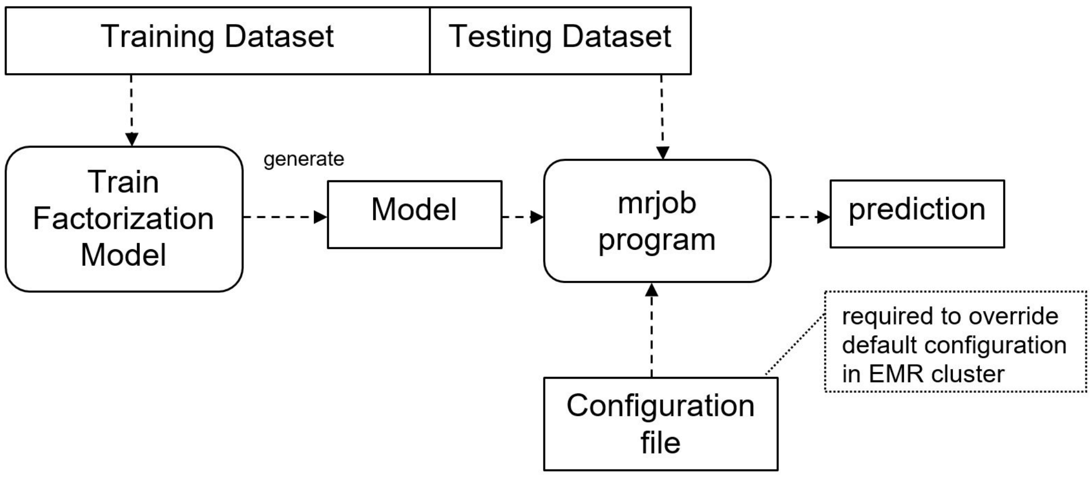


Figure 4.3: Workflow of mrjob Program

have applied five-fold cross-validation. We randomly split the dataset into five partitions (fold) of equal size. For each partition, we train the model with four sets and test on one set. We then calculate the test score for each partition and the final value is average of all the partitions. The Figure 4.4 shows the illustration of the five-fold cross-validation. We train and save the model several times using stochastic gradient descent optimizing algorithm. The saved model file consists of user bias, item bias, user factors, and item factors. The generated model file and testing set are used as input to the the mrjob program. The description of the input file required for mrjob program is further described in Figure 4.3.

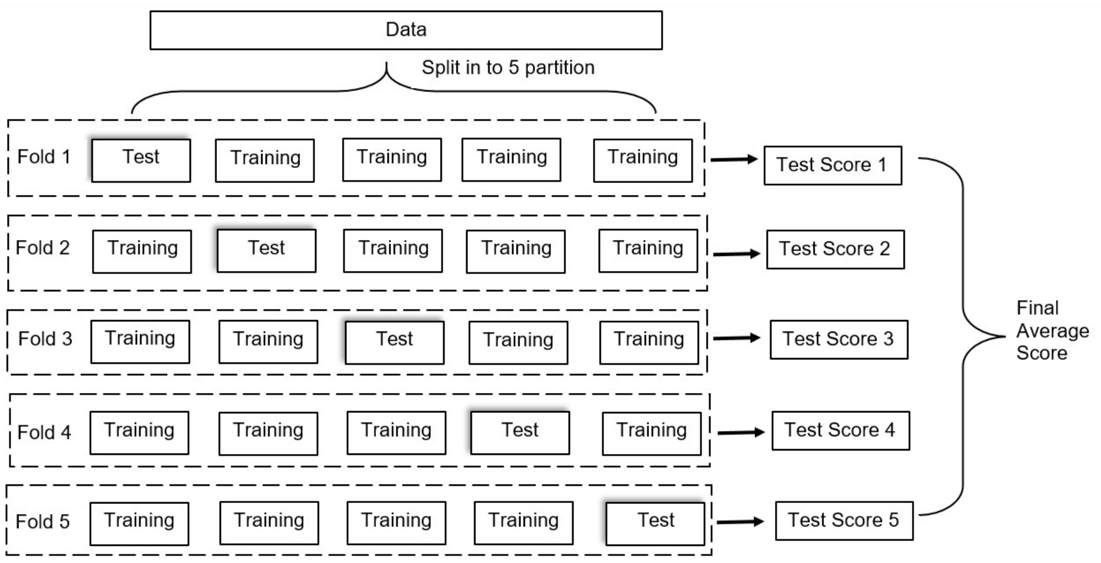


Figure 4.4: Schematic of Five-Fold Cross-Validation

4.2.1.2 Data Format

The mrjob program takes two input files, named test dataset and model file through stdin and runs multi-step *map* and *reduce* functions. A sample conversion of raw data into the input test dataset for mrjob program is displayed on the Figure 4.5. On other hand, the model file contains the learned parameters from the stochastic gradient descent algorithm. The parameters are user bias (ubias), item bias (ibias), user factors (ufactorid:ufactorvalue) and item factors (ifactorid:ifactorvalue). The data flow of the blend is illustrated in the Figure 4.5.

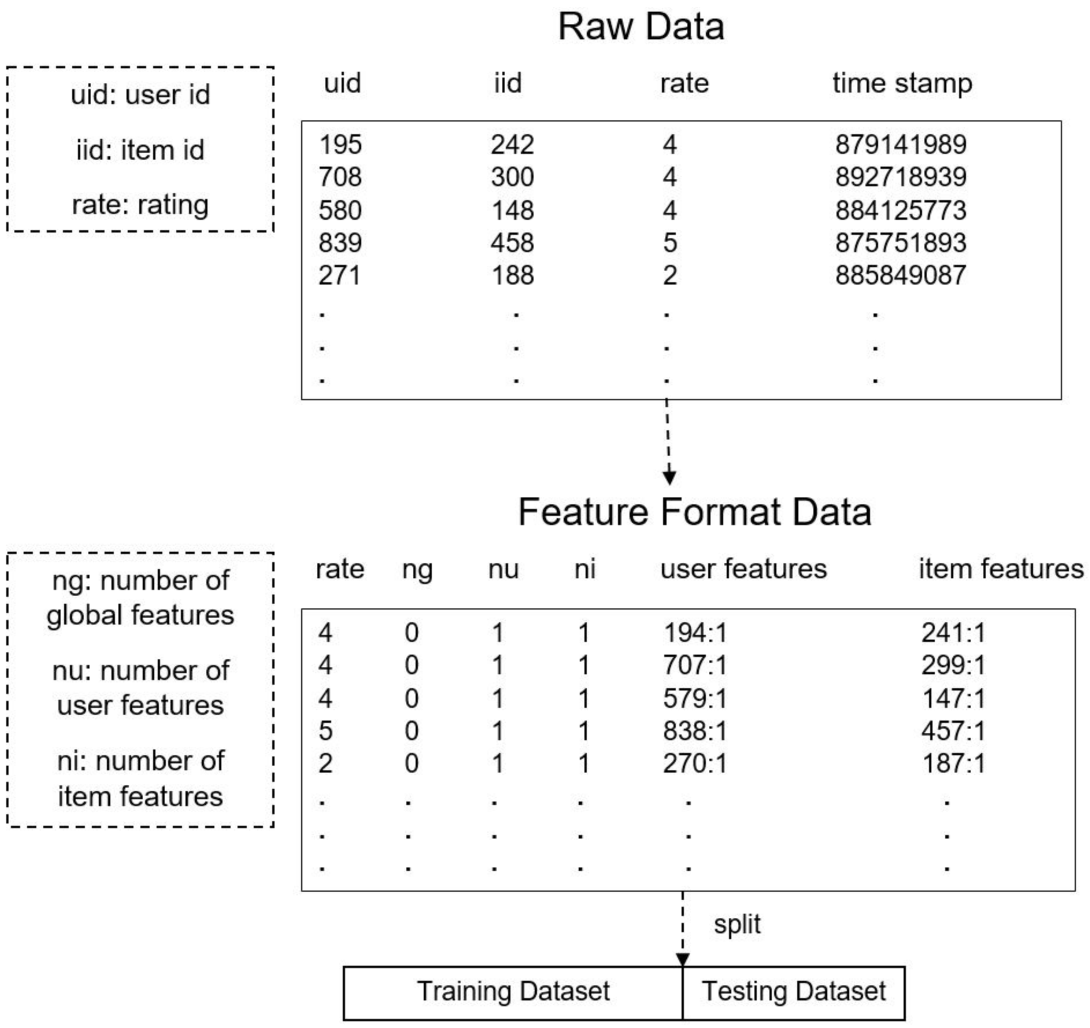


Figure 4.5: Data Format for mrjob

4.2.1.3 Mapping Phase

The map function starts by reading each line from the testing dataset file which is in the following format:

line = rate ng nu ni <global features> <user features> <item features>

where,

rate = prediction target

ng = number of nonzero global features

nu = number of nonzero user features

ni = number of nonzero item features

<global features> = gid[1]:gvalue[1] . . . gid[ng]:gval[ng]

<user features> = uid[1]:uvalue[1] . . . uid[nu]:uval[nu]

<item features> = iid[1]:ivalue[1] . . . iid[ni]:ival[ni]

The id and value in the feature format data correspond to the feature id and feature value of nonzero entity. As the *mapper* reads the testing set, it parses (uid, [0, uid, uvalue]) and (iid, [2, iid, ivalue]) as key-value pairs, where, uid and iid are keys and corresponding are it's values. Since, feature format data does not contain any global features, we neglect the gid and gvalue portion of data. Lets, consider the first map produced the following output:

(0,[0,0,1])

(1,[0,1,1])

Similarly, the second map produced:

(0,[2,0,1])

(1,[2,1,1])

where the key is either uid or iid and the corresponding values are stored in the list. Once the map function reaches end of file of test dataset, it starts reading the model file. The map function starts parsing model file as (uid, [1, uid, ubias]), (iid, [3, iid, ibias]), (ufactorid, [4, ufactorid, ufactorvalue]) and (ifactorid,[5, ifactorid, ifactorvalue]) as key-value pairs respectively. Let's consider the first mapper produced the following output:

(0,[1,0,-0.077])

(1,[1,1,0.548])

where, the key and values are in format of (uid,(1,uid,ubias))

Similarly, the second map produced:

(0,[3,0,0.250])

(1,[3,1,0.347])

where, the key and values are in format of (iid,(3,iid,ibias))

Likewise, the third map produced:

(0,[4,0,[0, 0, ...]])

(1,[4,1,[0, 0, ...]])

where, the key and values are in format of (ufactorid,(4,ufactorid,ufactorvalue))

Finally, the fourth map produced:

(0,[5,0,[0, 0, ...]])

(1,[5,1,[0, 0, ...]])

where the key and values are in format of (ifactorid,(5,ifactorid,ifactorvalue)).

The map function here is simple, we consider this as a data preparation phase, setting up the data in such a way that the combiner function can perform its work on it.

4.2.1.4 Shuffle and Sort : Map Phase

As the map function starts to emit its key-value pairs, a partition is created for each unique key. Within each partition, the framework performs an in-memory sort by keys and passes the output to combiner phase. The output of this phase would be collecting and sorting the map outputs. The sample shuffle and sort of the key-value pairs produced by map phase in previous phase would be:

(0,[[0,0,1], [2,0,1],[1,0,-0.077], [3,0,0.250], [4,0,[0, 0, ...]], [5,0,[0, 0, ...]]])

(1,[[0,1,1], [2,1,1], [1,1,0.548], [3,1,0.347], [4,1,[0, 0, ...]], [5,1,[0, 0, ...]]])

where, the shuffling and sorting is done based on the unique key.

4.2.1.5 Combiner Phase

The combiner is an optimization, it will run for each unique key present on the map output. Since, the combiner is reduce type function, we reduce the values of each unique key by computing the values as defined by the factorization model. The sample output of combiner phase would be:

(0,[-0.077,0.250,0])

(1,[0.548,0.347,0])

where, the values provided by shuffle and sort function for a unique key is considerably reduced to smaller values. The output of the combiner function is similar to the output of shuffle and sort function which is now transferred to reduce phase for final processing.

4.2.1.6 Shuffle and Sort : Reduce Phase

The sort function of reduce phase fetches the output from combiner and merges them into a larger sorted file. As the input to reduce comes from multiple map function, the sort phase merges the map outputs, maintaining their sort ordering.

4.2.1.7 Reduce Phase

During the reduce phase, the reduce function is invoked for each key in the sorted output. All the values of the same key are gathered together and the sum of the values are calculated by the reducer. We add baseline estimate value 3 to the sum of values of each unique key and write the new key-value pair to

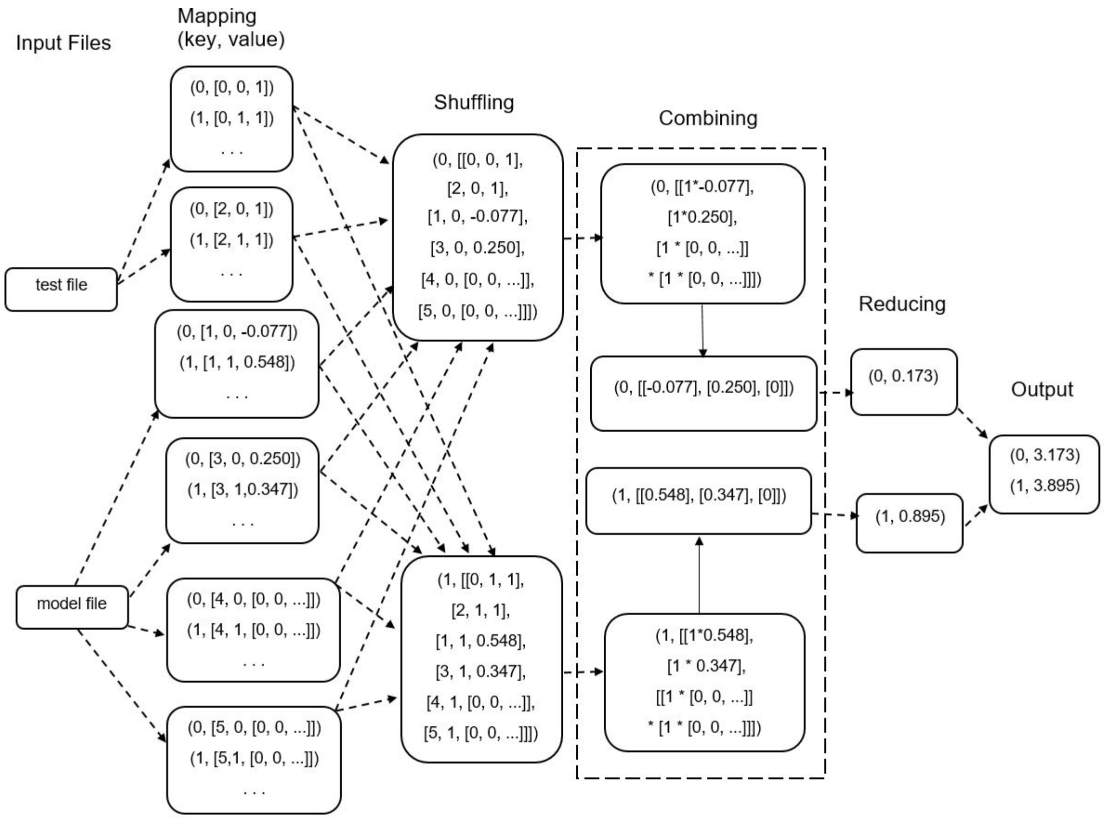


Figure 4.6: Factorization Model with MapReduce Framework

output file defined in stdout. The workflow of the MapReduce framework of factorization model is shown in the Figure 4.6.

4.2.2 Running mrjob

In this section, we explain the working mechanism of mrjob program that executes the factorization model. We start by analyzing the prerequisite to run the program and explore how the program works. We then test the mrjob program on the local machine, which by default, runs on a single python process that allows us to validate the functionality of our code. Finally, we deploy the program into Amazon EMR for distributed computing using multiple nodes.

4.2.2.1 Workflow

A MapReduce job usually splits the data into chunk which are then processed by the map tasks in parallel. The framework then sorts the output of the map tasks, which are then input to reduce task. The job of factorization model is defined by a class that inherits from mrjob.

```
class infer(MRJob):
    . . .
    . . .
if __name__ == "__main__":
    infer.run()
#where, infer is the job class.
```

This class contains methods that define the steps of our job. A step consists of a mapper, a combiner and a reducer. The mrjob program for factorization model uses two steps to complete the job, so, we override the default step method to return a list of MRSteps. We start by importing MRJob and MRStep.

```
from mrjob.job import MRJob

from mrjob.step import MRStep
```

Then, we define multiple step method inside the class by changing the default single step method as:

```
def steps( self ):
    return [
        MRStep(mapper=self.mapping_values, reducer=self.reducing_values), # First
            Step.
        MRStep(mapper=self.mapping_values_keys,
            reducer=self.sum_of_values_by_keys) # Second Step.
    ]
```

Here, in the first step, we have one mapper and one reducer. The mapper preprocesses the data and reducer perfoms calculations. Since, the mapper of the first step has to read two input files, we define a method that checks the input files and identifies them as a testing dataset file or a model file. Once, the identification has been done, the mapper starts to parse the data for reducer. The method that identifies the input file is shown next:

```
def configure_args(self):
    super(infer, self).configure_args()
     self.add_passthru_arg("--test-input", default="test", dest="test")
def decide_input_file(self):
    filename = jobconf_from_env("map.input.file")
    # this is to get the name of the file a mapper is reading input from.
     if self.options.test in filename:
    # return 1 if test.feature , return 2 if mapper is reading .model file.
            return 1
        else:
            return 2
```

After the first step's reducer run, the individual results are arbitrarily assigned to second step's mapper for further processing. The second mapper takes key-value pairs and sends them to second step's reducer for final processing. The second step's reducer sums all the values and adds baseline estimates (identified as base_score in next code snippet) for every unique key emitted by mapper. Finally, the reducer emits the key-value pair to the output file. The value we are expecting from the reducer are the predicted rating of a movie by a user. The mrjob allows us to write a multi step MapReduce program inside a single class which is easier to debug and is very efficient.

```
# second step mapper takes key-value pairs.
def mapping_values_keys(self, k, v):
    yield k, v
#seond step reducer sums all the values of and unique key and adds base_score to
     final value.
def sum_of_values_by_keys(self, k, values):
    yield k, sum(values) + base_score
```

4.2.2.2 Running Locally

Till here, we have a working MapReduce program written using mrjob library in python, the next step is to run the program locally on a test dataset. Among many flexibility delivered by mrjob library, one of them is that we can run the MapReduce program locally, without having Hadoop installed. While running the program locally, it runs on a single python process which is technically not a distributed computing. Once, we validate the functionality of the program locally, we then deploy the program to a cluster which will be covered in next section. We run the mrjob program using the below command:

```
python rating_pred_mrjob.py test_input model_input > output.txt
```

where rating_pred_mrjob.py is executable mrjob program written in python; test_input is the input testing dataset file and model_input is the model input file to mrjob program. The progress information displayed on Anaconda Prompt console after executing the command is as (some lines are arranged for clarity):

```
No configs found; falling back on auto-configuration

No configs specified for inline runner

Creating temp directory..

C:\Temp\rating_pred_mrjob.Nabin.20200304.034608.429581

Running step 1 of 2...

Running step 2 of 2...

job output is in ..

C:\Temp\rating_pred_mrjob.Nabin.20200304.034608.429581\output

Streaming final output from..

C:\Temp\rating_pred_mrjob.Nabin.20200304.034608.429581\output...

Removing temp directory..

C:\Temp\rating_pred_mrjob.Nabin.20200304.034608.429581...
```

The output will be written to output.txt and in cases where the output.txt file does not exist, the program creates the file and writes the output, and when the output.txt file pre-exists, the program rewrites the output on the text file.

4.2.2.3 Running on a Cluster using Amazon EMR

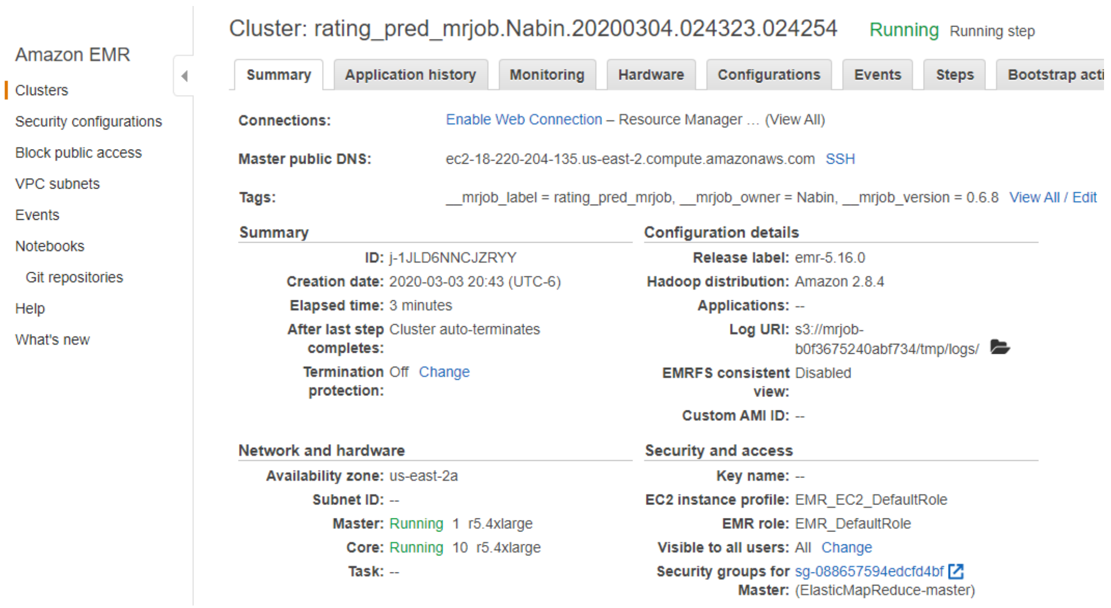
Since, the program executed successfully on local machine as shown in section 4.2.2.2, we now deploy the program on the full dataset on a Amazon EMR cluster. To launch the job in the distributed computing setting of Amazon EMR, we need to run the driver, specifying the cluster that we want to run the job on with the conf option. The conf file as initially shown in Figure 4.3, overrides the default configuration of EMR cluster setting giving us ability to control the flow of our program. The config files are interpreted as YAML, a human-readable data-serialization language commonly used for configuration files and in applications where data are being stored or transferred. The example of config file is as:

```
runners:
    emr:
        aws_access_key_id: xxxxxxxxxxxxxxxx
        aws_secret_access_key: xxxxxxxxxxxxxxxxx
        num_core_instances: 10
        master_instance_type: r5.4xlarge
        core_instance_type: r5.4xlarge
        region: us-east-2
```

where aws_access_key_id and aws_secret_access_key are the security credentials of user's Amazon Web Service (AWS). The security credentials are required to verify the user's identity and to check if the user has permission to access the requested resources. The num_core_instances are the number of child nodes or instances, we want to use in EMR cluster. The master_instance_type and core_instance_type are the type of Amazon Elastic Compute Cloud (EC2) the user wants to use. The region is Amazon data-center geographical area. To run the mrjob program into Amazon EMR cluster, we use the below command:

```
python rating_pred_mrjob.py -r emr –conf-path=mrjob.conf test_input
model_input > output.txt
```

where rating_pred_mrjob.py is executable mrjob program written in python; test_input is the input testing dataset file and model_input is the model input file to mrjob program. The r is abbreviaton for runner, which communicates with the framework to run the program on EMR cluster. The conf-path is the path of the configuration file which contains the setting for EMR cluster and the output of the program is written into stdout which is output.txt. After the execution of the command, the job launches on EMR cluster and polls for progress, writing a line summmarizing the map and reduce's progress whenever



Cluster: rating_pred_mrjob.Nabin.20200304.024323.024254 Running Running step

Connections:	Enable Web Connection – Resource Manager ... (View All)
Master public DNS:	ec2-18-220-204-135.us-east-2.compute.amazonaws.com SSH
Tags:	__mrjob_label = rating_pred_mrjob, __mrjob_owner = Nabin, __mrjob_version = 0.6.8 View All / Edit

Summary

ID:	j-1JLD6NNCJZRYY
Creation date:	2020-03-03 20:43 (UTC-6)
Elapsed time:	3 minutes
After last step completes:	Cluster auto-terminates
Termination protection:	Off Change

Configuration details

Release label:	emr-5.16.0
Hadoop distribution:	Amazon 2.8.4
Applications:	--
Log URI:	s3://mrjob-b0f3675240abf734/tmp/logs/
EMRFS consistent view:	Disabled
Custom AMI ID:	--

Network and hardware

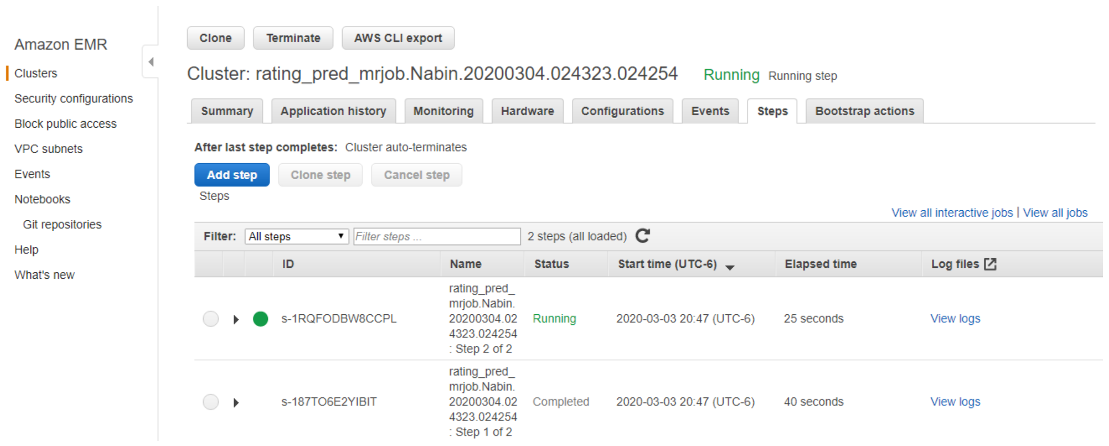
Availability zone:	us-east-2a
Subnet ID:	--
Master:	Running 1 r5.4xlarge
Core:	Running 10 r5.4xlarge
Task:	--

Security and access

Key name:	--
EC2 instance profile:	EMR_EC2_DefaultRole
EMR role:	EMR_DefaultRole
Visible to all users:	All Change
Security groups for Master:	sg-088657594edcfd4bf (ElasticMapReduce-master)

Figure 4.7: Overview of mrjob in EMR Web User Interface

either changes. The progress information displayed on Anaconda Prompt console after executing the command. As the job executes on the EMR cluster, the input files and the mrjob program is first copied to Amazon Simple Storage Service (Amazon S3) bucket. S3 bucket keeps the input files, generated logs and executable files with them while the EMR cluster runs the mrjob program. The Figure 4.7 shows information of cluster running ten core instances on the EMR cluster to processes the map and reduce task parallelly. The final output is streamed from EMR S3 bucket to path defined on stdout. The steps in EMR are shown in Steps tab of cluster. The Figure 4.8 shows that there are two steps running in EMR cluster. We record the elasped time of both steps to calculate



Cluster: rating_pred_mrjob.Nabin.20200304.024323.024254 Running Running step

After last step completes: Cluster auto-terminates

2 steps (all loaded)

ID	Name	Status	Start time (UTC-6)	Elapsed time	Log files
s-1RQFODBW8CCPL	rating_pred_mrjob.Nabin.20200304.024323.024254 : Step 2 of 2	Running	2020-03-03 20:47 (UTC-6)	25 seconds	View logs
s-187TO6E2YIBIT	rating_pred_mrjob.Nabin.20200304.024323.024254 : Step 1 of 2	Completed	2020-03-03 20:47 (UTC-6)	40 seconds	View logs

Figure 4.8: Steps status of mrjob in EMR Web User Interface

the processing speed of large data. The experiment of analyzing elasped time using multiple core instances for different dataset are explained in next chapter.

CHAPTER 5
EXPERIMENTS

This chapter briefly evaluates the performance and accuracy of the suggested model. We design and run experiments on a variety of datasets to examine the effectiveness of the factorization model with the MapReduce framework. We start with defining the research question as:

5.1 Research Questions

The overall gist of this work is to develop a model which can integrate different dynamic models to generate accurate rating and process them using the MapReduce framework. Therefore, we perform the experiments aiming to answer the following research questions:

- RQ1 : Does the suggested feature-based factorization model have prediction accuracy on a large sparse dataset?
- RQ2 : How does the performance of the model vary across the different sizes of datasets?

To find answers to the above research questions, we start by describing the dataset and the experimental setup along with the evaluation criteria.

5.2 Evaluation Criteria

We evaluate the experiments using two methods - prediction accuracy and processing speed. For RQ2, the model uses the MapReduce framework to generate the rating of an item. As the model processes the large dataset using distributed computing, we measure the elapsed time for both step one and step two with varying instances. We use one instance, five instances and ten instances with the same hardware configuration for parallel processing of the large user-item dataset. For each computation with varying instances, we keep track of the processing time of mrjob steps.

For RQ1, we use root mean square error (RMSE) as described in section 3.7 to measure the prediction accuracy. Root mean square formula as presented on equation 3.16, measures the error between the predicted value and the corresponding known value. A lower RMSE value corresponds to the higher prediction accuracy. In all the experiments henceforth, the reported RMSE values are measured on all the five-fold's testing and training dataset. For a model to be considered as a good model, the RMSE of training dataset and test dataset should be similar. If the RMSE of the testing dataset is much higher than the training dataset, then we have overfit the data into the model, i.e., the model generated accurate prediction on the seen dataset but generates inaccurate prediction on an unseen dataset. We take RMSE of each fold and find the average of folds for a dataset.

5.3 Experimental Design and Analysis

This experiment is designed to solve the linear regression problem of equation 3.15. The goal of training a model is to discover the set of weights and biases that have low loss. The regression model that we examine here uses a loss function called squared loss (L_2 loss) as shown in equation 3.17. The iterative method as described in Algorithm 1, updates the model's parameters repeatedly until we reach the lowest loss value. Here, in this experiment, we iterate the algorithm until the overall loss stops changing or at least changes very slowly. When that happens, we called the model has converged. The overall loss is measured using root mean square error (RMSE) formula of equation 3.16. To accurately check if our model has converged, we keep track of the RMSE value of different model and pick the one with low RMSE. We start this section with the dataset we use for the experiments, experiment procedure and finally the experiments on datasets.

5.3.1 Dataset

We use publicly available MovieLens GroupLensResearch (2020) dataset to evaluate our models. The datasets are collected over various periods of time,

Datasets	Users	Items	Ratings	Density %
ML100K	943	1,682	4,100,000	28.491
ML1M	6,040	3,952	1,000,209	4.19
ML10M	71,567	10,681	10,000,054	1.30
ML25M	162,541	62,423	25,000,095	0.246

"Density" refers to density of each dataset, calculated as (Density = Ratings/Users $\times$ Items)

Table 5.1: MovieLens Datasets

depending on the size of the datasets. We use four datasets for experiments, i.e., MovieLens 100K Dataset (ML100K), MovieLens 1M Dataset (ML1M), MovieLens 10M Dataset (ML10M), and MovieLens 25M Dataset (ML25M). We do not pre-process the MovieLens datasets before using into the model as its already filtered. The description of datasets is described in Table 5.1. We calculate the density of datasets to see how sparse the datasets are and as the dataset size increases, the sparsity increases; this can be seen in Table 5.1. Each dataset is further divided into different sets by five-fold cross-validation as shown in Figure 4.4, using a shell script that internally runs a Perl script. The script divides the MovieLens datasets into five folds with 80% and 20% splits on each fold, which is training and test dataset respectively.

5.3.2 Experimental Procedure

A overview of experimental procedure is described stepwise as following:

- Step 1 : Using the five-fold cross-validation technique as shown in Figure 4.4, we split the dataset into training and test datasets

- Step 2 : We randomly shuffle the training dataset using random seed value as 10

- Step 3 : We then convert the raw training and test datasets into feature-based data as shown in Figure 4.5

- Step 4 : The training program runs the factorization model of equation 3.19 on feature-based training dataset produced by Step 3

- Step 5 : Once, Step 4 is completed, we calculate the loss function using the Formula 3.17

- Step 6 : We then run Stochastic Gradient Descent using the Algorithm 1 to reduce the loss function

- Step 7 : We save the optimized model generated by Step 6 into local machine's hard drive for each cycle and repeat Step 4 till Step 6 until the training round is over (number of training rounds are provided by the user in a configuration file)

- Step 8 : We calculate the RMSE value using formula 3.16 for models.

- Step 9 : We take the n^{th} trained model having low RMSE, and test dataset as input to the mrjob program as shown in Figure 4.3

- Step 10 : We deploy the mrjob program into Amazon Elastic MapReduce (EMR) cluster with multiple instances

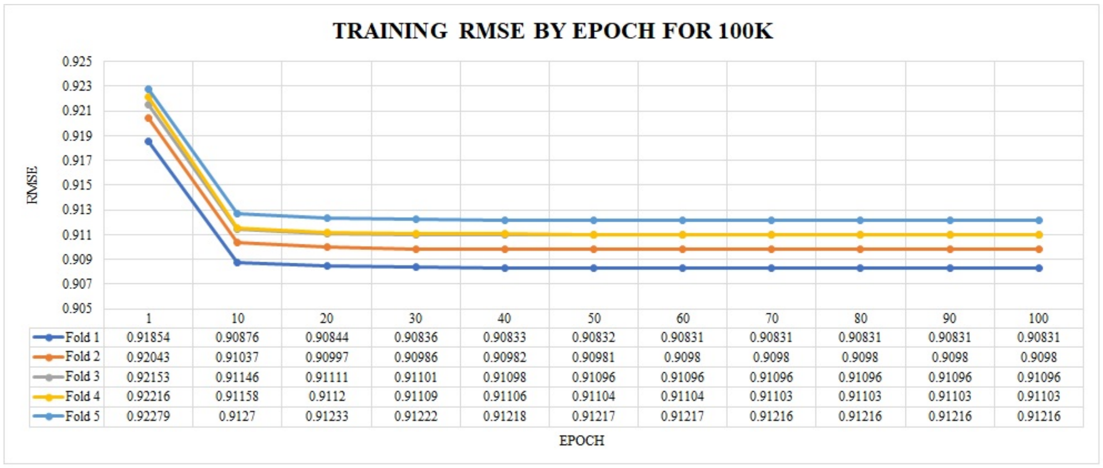
- Step 11 : The final predicted output is streamed from Amazon Simple Storage Service (S3) to the local machine's hard drive by Amazon Elastic MapReduce (EMR) cluster automatically

- Step 12 : Finally, we record the elapsed time for each mrjob steps computed in EMR

The factorization model of equation 3.19 is trained using learning rate (η) as 0.005 and regularization constant (λ) for item and user as 0.004.

5.3.3 Experiment : ML100K

The ML100K dataset consists of 100,000 ratings (1-5) from 943 users on 1,682 movies. Each user in the dataset has rated at least 20 movies. The data was collected through the MovieLens website GroupLensResearch (2020) during the seven months period from September 19, 1997 through April 22, 1998. Being the oldest dataset, these dataset has already been cleaned and is ready to be used for experiments. We partition the dataset into five-folds and train the model using the training dataset. As the models becomes available after the training phase, we compute the RMSE to find the converged model. The RMSE of models are shown in Figure 5.1. Analyzing the Figure 5.1, we find the lowest average RMSE is achieved by the 70th model, which means that our model has converged here and its values are presented in Table 5.2. The average lowest RMSE value obtained by the factorization model for the train dataset is 0.91045. We then use the 70th model and test dataset as an input to the mrjob program and deploy the mrjob program into EMR. The elapsed time for each fold to complete the testing phase computation is shown in Table 5.4 and pictorial representation is shown in Figure 5.2. The seconds are the sum of



	1	10	20	30	40	50	60	70	80	90	100
Fold 1	0.91854	0.90876	0.90844	0.90836	0.90833	0.90832	0.90831	0.90831	0.90831	0.90831	0.90831
Fold 2	0.92043	0.91037	0.90997	0.90986	0.90982	0.90981	0.9098	0.9098	0.9098	0.9098	0.9098
Fold 3	0.92153	0.91146	0.91111	0.91101	0.91098	0.91096	0.91096	0.91096	0.91096	0.91096	0.91096
Fold 4	0.92216	0.91158	0.9112	0.91109	0.91106	0.91104	0.91104	0.91103	0.91103	0.91103	0.91103
Fold 5	0.92279	0.9127	0.91233	0.91222	0.91218	0.91217	0.91217	0.91216	0.91216	0.91216	0.91216

Figure 5.1: RMSE by Epoch for ML100K five-fold

both step one and step two of the mrjob program. We use m5.xlarge Amazon EC2 instances for Amazon EMR clusters on both master and core nodes. Table 5.3 shows the specification of m5.xlarge.

RMSE	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Test Dataset	0.9538	0.9442	0.9395	0.9391	0.9372	0.94276
Train Dataset	0.90831	0.9098	0.91096	0.91103	0.91216	0.91045

Table 5.2: RMSE of 100K five-fold

Instance	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)
m5.xlarge	4	16	EBS-Only	Up to 10

Table 5.3: Configuration of EMR Instances for ML100K, ML1M, and ML10M

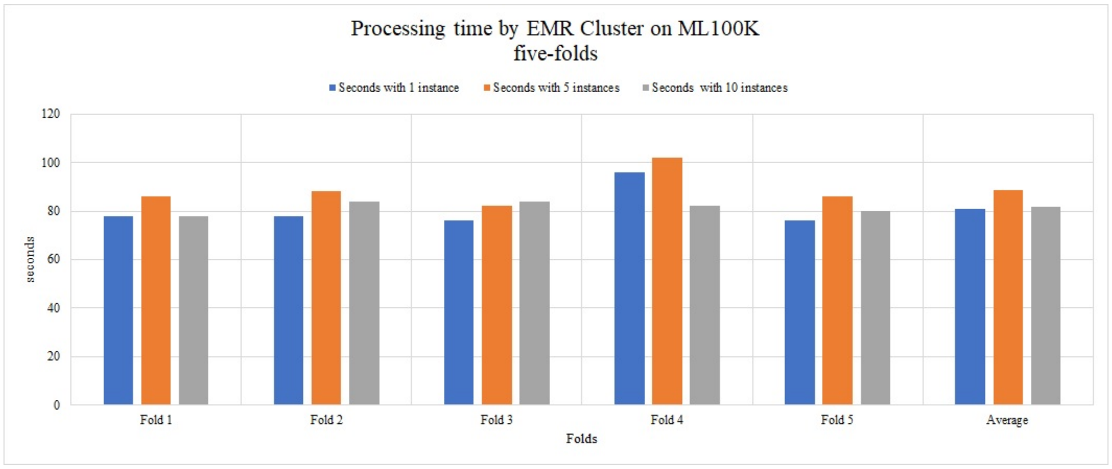


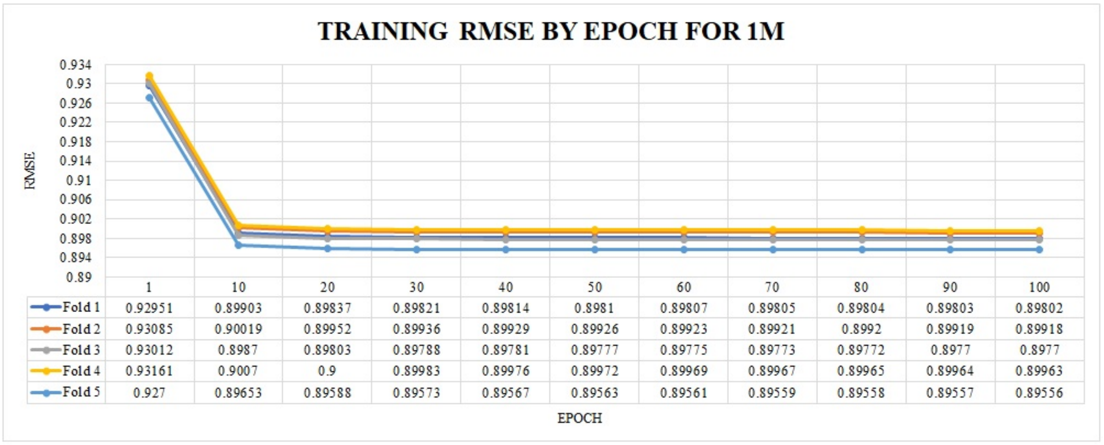
Figure 5.2: Processing time for ML100K five-fold in EMR

EMR Cluster	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Seconds with 1 instance	78	78	76	96	76	80.8
Seconds with 5 instances	86	88	82	102	86	88.8
Seconds with 10 instances	78	84	84	82	80	81.6

Table 5.4: Processing time for 100K five-fold in EMR

5.3.4 Experiment : ML1M

The ML1M dataset consists of 1,000,209 ratings (1-5) from approximately 3,900 users on 6,040 movies. Each user in the dataset has rated at least 20 movies. The data was collected through the MovieLens website GroupLensResearch (2020) from the year 2000 through 2003. The dataset has already been cleaned and is ready to be used for experiments. We partition the dataset into five-folds and train the model using the training dataset. As the model becomes available after the training phase, we compute the RMSE to find the converged model.



	1	10	20	30	40	50	60	70	80	90	100
Fold 1	0.92951	0.89903	0.89837	0.89821	0.89814	0.8981	0.89807	0.89805	0.89804	0.89803	0.89802
Fold 2	0.93085	0.90019	0.89952	0.89936	0.89929	0.89926	0.89923	0.89921	0.8992	0.89919	0.89918
Fold 3	0.93012	0.8987	0.89803	0.89788	0.89781	0.89777	0.89775	0.89773	0.89772	0.8977	0.8977
Fold 4	0.93161	0.9007	0.9	0.89983	0.89976	0.89972	0.89969	0.89967	0.89965	0.89964	0.89963
Fold 5	0.927	0.89653	0.89588	0.89573	0.89567	0.89563	0.89561	0.89559	0.89558	0.89557	0.89556

Figure 5.3: RMSE By Epoch for ML1M five-fold

The RMSE of models are shown in Figure 5.3. Analyzing the Figure 5.3, we find the lowest average RMSE is achieved by the 100^{th} model, which means that our model has converged here and its values are presented in Table 5.5. The average lowest RMSE value obtained by the factorization model for train dataset is 0.898018. We then use the 100^{th} model and test dataset as input to the mrjob program and deploy the mrjob program into EMR. The elapsed time for each fold to complete the testing phase computation is shown in Table 5.6 and pictorial representation is shown in Figure 5.4. The seconds are the sum of both step one and step two of the mrjob program. We use m5.xlarge Amazon EC2 instances for Amazon EMR clusters on both master and core nodes. The configuration of m5.xlarge Amazon EC2 instances is shown in Table 5.3.

RMSE	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Test Dataset	0.98549	0.98206	0.98581	0.984	0.98829	0.98513
Train Dataset	0.89802	0.89918	0.8977	0.89963	0.89556	0.898018

Table 5.5: RMSE of 1M five-fold

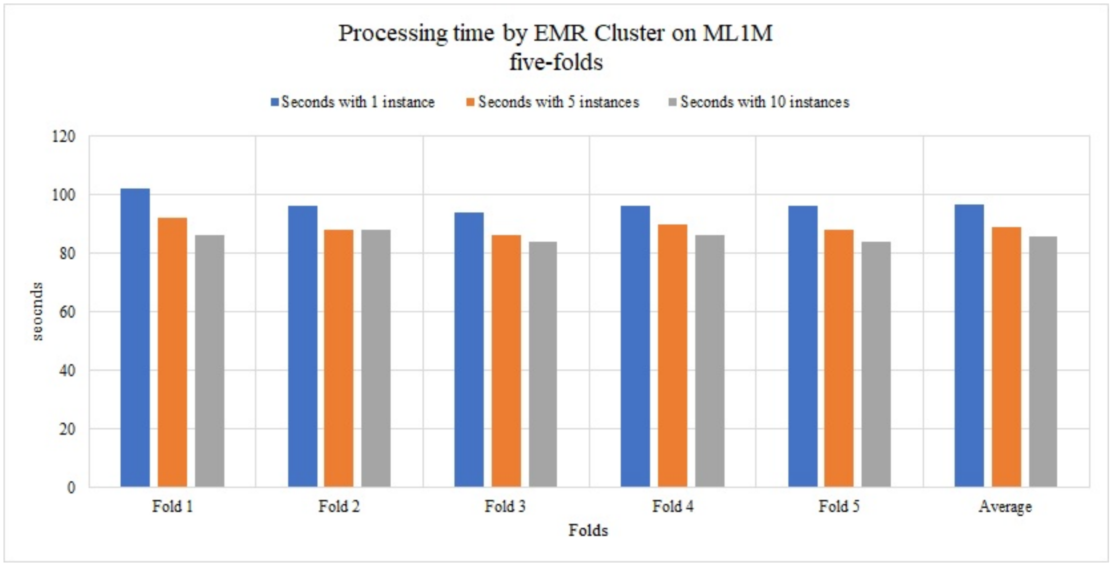


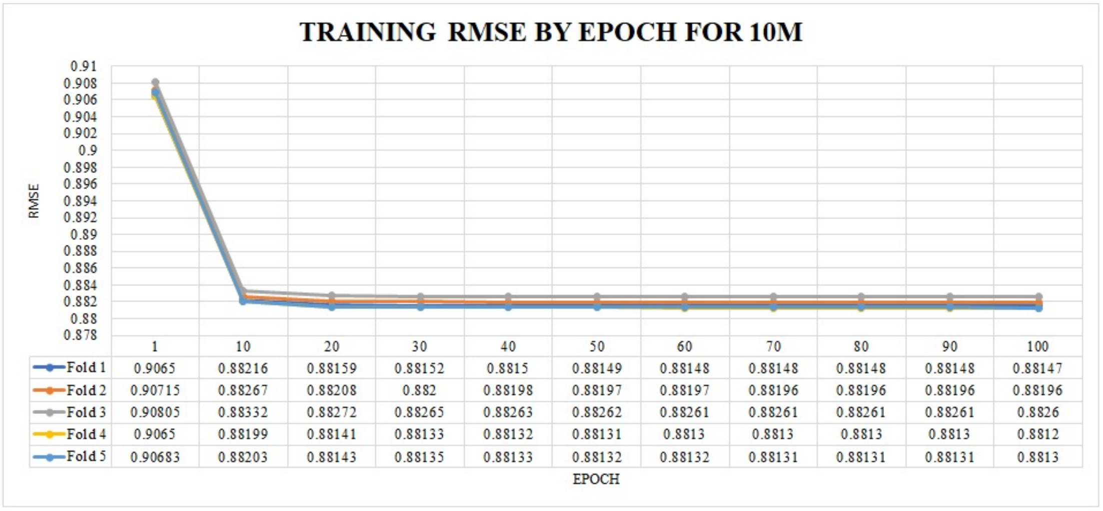
Figure 5.4: Processing time for ML1M five-fold in EMR

EMR Cluster	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Seconds with 1 instance	102	96	94	96	96	96.8
Seconds with 5 instances	92	88	86	90	88	88.8
Seconds with 10 instances	86	88	84	86	84	85.6

Table 5.6: Processing time for ML1M five-fold in EMR

5.3.5 Experiment : ML10M

The ML10M dataset consists of 10,000,054 ratings (1-5) from approximately 71,567 users on 10,681 movies. Each user in the dataset has rated at least 20



	1	10	20	30	40	50	60	70	80	90	100
Fold 1	0.9065	0.88216	0.88159	0.88152	0.8815	0.88149	0.88148	0.88148	0.88148	0.88148	0.88147
Fold 2	0.90715	0.88267	0.88208	0.882	0.88198	0.88197	0.88197	0.88196	0.88196	0.88196	0.88196
Fold 3	0.90805	0.88332	0.88272	0.88265	0.88263	0.88262	0.88261	0.88261	0.88261	0.88261	0.8826
Fold 4	0.9065	0.88199	0.88141	0.88133	0.88132	0.88131	0.8813	0.8813	0.8813	0.8813	0.8812
Fold 5	0.90683	0.88203	0.88143	0.88135	0.88133	0.88132	0.88132	0.88131	0.88131	0.88131	0.8813

Figure 5.5: RMSE By Epoch for ML10M five-fold

RMSE	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Test Dataset	0.98974	0.99029	0.98629	0.987407	0.98357	0.987459
Train Dataset	0.88147	0.88196	0.8826	0.8812	0.8813	0.881706

Table 5.7: RMSE of 10M five-fold

movies. The data was released by MovieLens GroupLensResearch (2020) in the year 2009. The dataset has already been cleaned and is ready to be used for experiments. We partition the dataset into five-folds and train the model using the training dataset. As the model becomes available after the training phase, we compute the RMSE to find the converged model. The RMSE of models are shown in Figure 5.5. Analyzing the Figure 5.5, we find the lowest average RMSE is achieved by the 100^{th} model, which means that our model has converged here and its values are presented in Table 5.7. The average lowest RMSE value

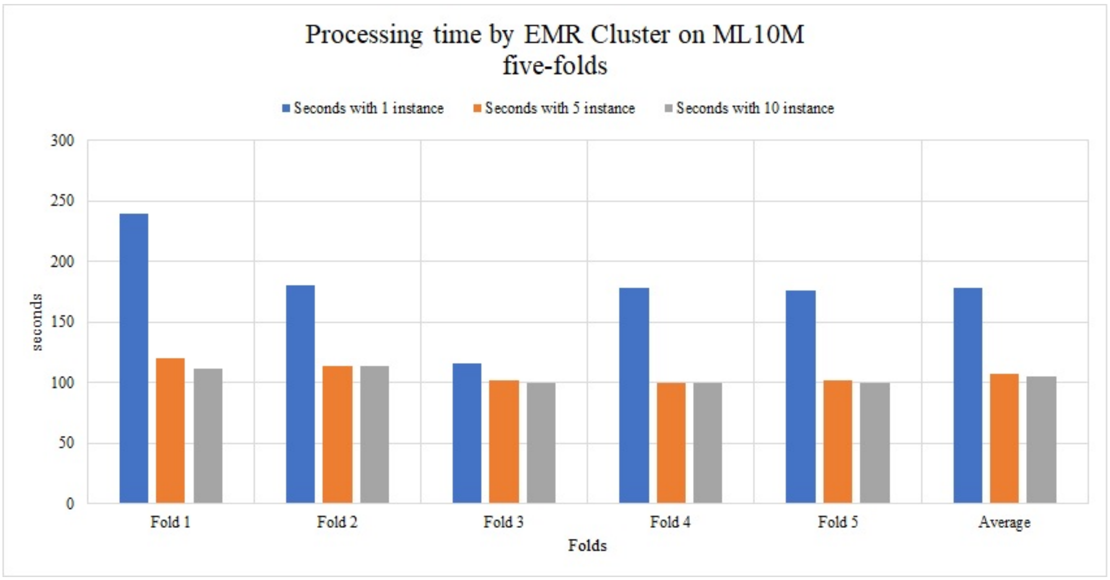


Figure 5.6: Processing time for ML10M five-fold in EMR

obtained by the factorization model for train dataset is 0.881706. We then use the 100^{th} model and test dataset as input to the mrjob program and deploy the mrjob program into EMR. The elapsed time for each fold to complete the testing phase computation is shown in Table 5.8 and pictorial representation is shown in Figure 5.6. The seconds are the sum of both step one and step two of the mrjob program. We use m5.xlarge Amazon EC2 instances for Amazon EMR clusters on both master and core nodes. The configuration of m5.xlarge Amazon EC2 instances is shown in Table 5.3.

EMR Cluster	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Seconds with 1 instance	240	180	116	178	176	178
Seconds with 5 instances	120	114	102	100	102	107.6
Seconds with 10 instances	112	114	100	100	100	105.2

Table 5.8: Processing time for ML10M five-fold in EMR

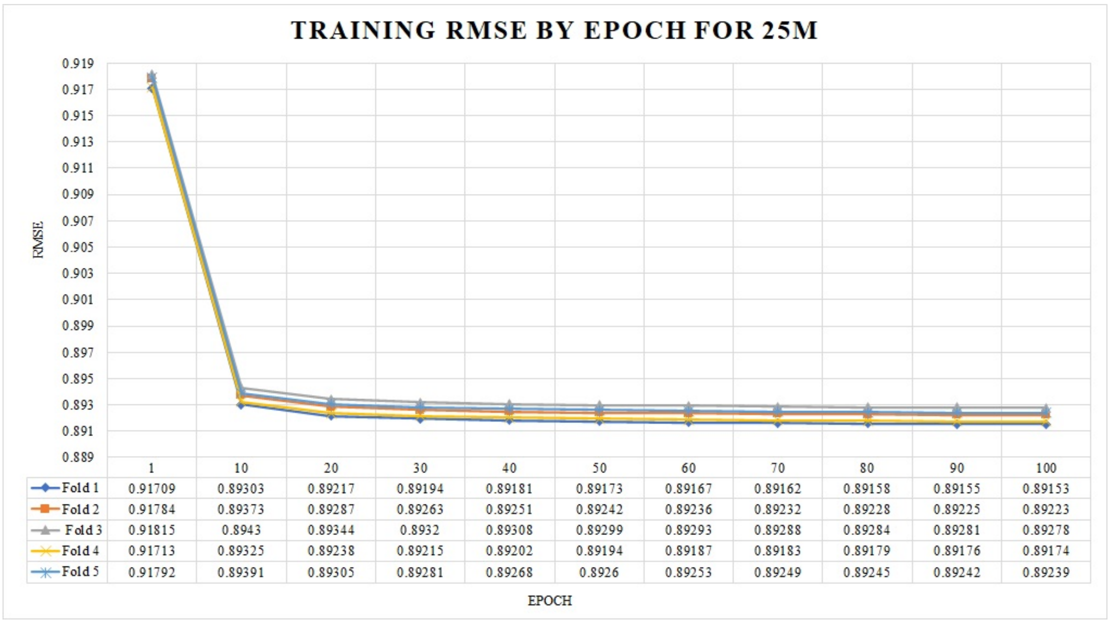
5.3.6 Experiment : ML25M

The ML25M dataset consists of 25,000,095 ratings (1-5) from approximately 162,541 users on 62,423 movies. Each user in the dataset has rated at least 20 movies. The data was created between January 09, 1995 and November 21, 2019. The data was released by MovieLens on November 21, 2019. The dataset has already been cleaned and is ready to be used for experiments. We partition

RMSE	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Test Dataset	1.018288	1.01734	1.01511	1.017653	1.012659	1.01621
Train Dataset	0.89153	0.89223	0.89278	0.89174	0.89239	0.89213

Table 5.9: RMSE of 25M five-fold

the dataset into five-folds and train the model using the training dataset. As the model becomes available after the training phase, we compute the RMSE to find the converged model. The RMSE of models are shown in Figure 5.7. Analyzing the Figure 5.7, we find the lowest average RMSE is achieved by the 100^{th} model, which means that our model has converged here and its values



	1	10	20	30	40	50	60	70	80	90	100
Fold 1	0.91709	0.89303	0.89217	0.89194	0.89181	0.89173	0.89167	0.89162	0.89158	0.89155	0.89153
Fold 2	0.91784	0.89373	0.89287	0.89263	0.89251	0.89242	0.89236	0.89232	0.89228	0.89225	0.89223
Fold 3	0.91815	0.8943	0.89344	0.8932	0.89308	0.89299	0.89293	0.89288	0.89284	0.89281	0.89278
Fold 4	0.91713	0.89325	0.89238	0.89215	0.89202	0.89194	0.89187	0.89183	0.89179	0.89176	0.89174
Fold 5	0.91792	0.89391	0.89305	0.89281	0.89268	0.8926	0.89253	0.89249	0.89245	0.89242	0.89239

Figure 5.7: RMSE By Epoch for ML25M five-fold

are presented in Table 5.9. The average lowest RMSE value obtained by the factorization model for train dataset is 0.89213. We then use the 100^{tb} model and test dataset as input to the mrjob program and deploy the mrjob program into EMR. The elapsed time for each fold to complete the testing phase computation is shown in Table 5.11 and pictorial representation is shown in Figure 5.6. The seconds in Table 5.11 is the sum of both steps one and two of the mrjob program. We use r5.4xlarge Amazon EC2 instance for Amazon EMR cluster for 25 million dataset on both master and core nodes. The specification of m5.xlarge instance is shown in Table 5.10.

Instance	vCPU	Memory (GiB)	Instance Storage (GiB)	Network Bandwidth (Gbps)
r5.4xlarge	16	128	EBS-Only	Up to 10

Table 5.10: Configuration of EMR Instances for ML25M

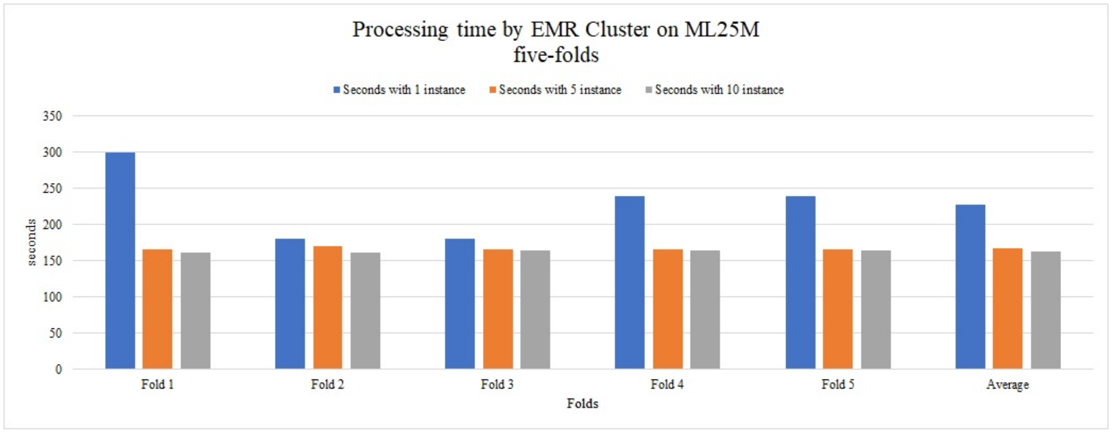


Figure 5.8: Processing time for ML25M five-fold in EMR

EMR Cluster	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
Seconds with 1 instance	300	180	180	240	240	228
Seconds with 5 instances	166	170	166	166	166	166.8
Seconds with 10 instances	162	162	164	164	164	163.2

Table 5.11: Processing time for ML25M five-fold in EMR

5.3.7 Overall Performance

The overall performance of MapReduce framework can be summarized in the bar chart of Figure 5.9. As illustrated by Figure 5.9, we see the time required to process large dataset by MapReduce is reduced by using multiple nodes. As the data size increases, it is better to use multiple node to process

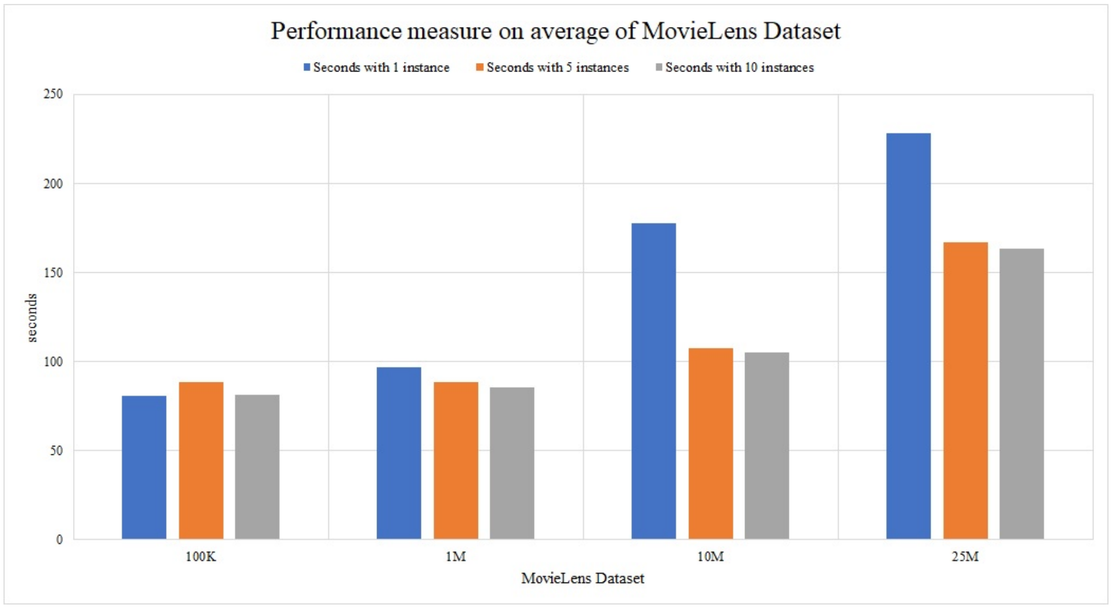


Figure 5.9: Performance measure of MovieLens Dataset on EMR

the data for speed and scalability. We also note that the factorization model does not need intense training rounds to achieve the lowest RMSE score even with the high sparse MovieLens 25 million dataset.

Average RMSE	ML100K	ML1M	ML10M	ML25M
Test Dataset	0.94276	0.98513	0.987459	1.01621
Train Dataset	0.91045	0.898018	0.881706	0.89213

Table 5.12: Average RMSE of MovieLens Dataset

5.4 Results and Discussion

Analyzing the above experiments of MovieLens dataset, we find that the performance of the factorization model is faster by using ten instances in

comparison to using one instance. This is because the computation of the rating is distributed across multiple nodes to process the large user and item data parallelly. We also note that the processing time using ten instances is less for ML25M than ML100K. This means that as the size of data increases, increasing the instances reduces the processing time as the data is processed parallelly.
The RMSE obtained on each fold is on the special case of the factorization model i.e. matrix factorization model. If we compare the test RMSE and train RMSE, we find both the RMSE are similar which means that our model is good as it neither overfits the data nor underfits the data. The RMSE acquired of the model is based on basic matrix factorization model and integrating neighborhood methods can result in lower RMSE score. The source code of the model and the mrjob program with instructions are provided in the GitHub repository (Giri, 2020b).

CHAPTER 6
CONCLUSION AND FUTURE WORKS

A personalized recommendation system has become an integral part of our daily lives. We rely heavily on these systems to provide us personalized suggestions as we read the news, watch videos, purchase products through online stores. These kinds of systems increase the decision-making capacity for users and in turn increases the product sales of an online merchandise. However, these kinds of systems suffer from large user-item data and inaccurate prediction problems. This thesis focuses to overcome the problem of large user-item data and the dilemma of different algorithms for accurate prediction. In this thesis, we used a collaborative filtering technique to build a recommendation system. We introduced a feature-based factorization model that helps to incorporate various dynamic models to generate an accurate prediction and uses the MapReduce framework to process large user-item data in a distributed manner.

The quality of a recommendation system can be expressed through various quantifiable factors including accuracy, computation efficiency, and choosing the top K value for neighborhood methods. Our work on the recommendation system uses relatively easy to measure quantifiable factors: accuracy - to measure the performance of the recommendation system and computation efficiency - to measure the processing time of the user-item data.

In the presented work, we used a special case of feature-based factorization model i.e. matrix factorization to generate a prediction. The prediction is evaluated in terms of accuracy by using root mean square error. In the immediate future work, we will perform experiments on the factorization model by integrating the neighborhood method as mentioned in Chapter 3. We will demonstrate the hypothesis that the prediction accuracy of the model increases as we integrate the neighborhood model.

Furthermore, we acknowledge the fact that explicit feedback (ratings) are not always available. We can collect the information from users through various other indirect ways such as a user's comment on the movie or the browsing history of the user or the amount of time spent by a user on a specific web page. All this information can be summarized and entitled under implicit feedback of the user. Another element that aids in generating accurate predictions is to acknowledge the drifting behavior of the user. A drifting behavior of the user can be a shift of user towards more family drama movie from adventure/action genre. An intelligent recommendation system needs to track these kinds of behavior to generate the best prediction for the user.

Furthermore, in a study presented by Netflix (Netflix Technology Blog, 2016), we learn that the artwork on the poster of the movie also plays a surprisingly significant role in the choices of the user. In our future work, we will focus on ways to capture and implement various user behavior patterns into the global features of the factorization model and process them simultaneously.

We also performed experiments using MapReduce framework to determine the increase in speed and efficiency when presented a large user-item data. The framework is relatively easy to implement and the results are also impressive, as shown on the experiments of chapter 4. The experiments performed on the thesis are on the available largest MovieLens dataset of 25 Million ratings of users. In the future work, we will perform the experiments on much larger dataset and on different variety of dataset apart from movie domain.

Last but not the least, with the limited time, the available computing resources and datasets, only few experiments could be performed and demonstrated in this thesis. We expect more experiments on large datasets with powerful computation resources needed for intense training of models, the parallelization framework should improve the efficiency of the recommendation system.

REFERENCES

Bell, R., Koren, Y., and Volinsky, C. (2007a). Modeling relationships at multiple scales to improve accuracy of large recommender systems. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’07, page 95–104, New York, NY, USA. Association for Computing Machinery,”https://doi.org/10.1145/1281192.1281206”.

Bell, R., Koren, Y., and Volinsky, C. (2007b). Modeling relationships at multiple scales to improve accuracy of large recommender systems. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’07, page 95–104, New York, NY, USA. Association for Computing Machinery, ”https://doi.org/10.1145/1281192.1281206”.

Bell, R. M. and Koren, Y. (2007a). Improved neighborhood-based collaborative filtering. In 1st KDDCup’07, San Jose, California.

Bell, R. M. and Koren, Y. (2007b). Lessons from the netflix prize challenge. SIGKDD Explor. Newsl., ”doi.org/10.1145/1345448.1345465”, 9(2):75–79.

Bell, R. M. and Koren, Y. (2007c). Scalable collaborative filtering with jointly. In ICDM, pages 43–52. IEEE Computer Society, "http://doi.ieeecomputersociety.org/10.1109/ICDM.2007.90".

Bennett, J. and Lanning, S. (2007a). The netflix prize. In Proceedings of the KDD Cup Workshop 2007, "http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf", pages 3–6, New York. ACM.

Bennett, J. and Lanning, S. (2007b). The netflix prize. In Proceedings of the KDD Cup Workshop 2007, pages 3–6, New York. ACM,"http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf".

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In Proceedings of KDD cup and workshop, volume 2007, page 35. Citeseer.

Billsus, D., Pazzani, M. J., et al. (1998). Learning collaborative information filters. In Icml, volume 98, pages 46–54.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. J. Mach. Learn. Res., 3(null):993–1022.

Breese, J., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, pages 43–52.

Canny, J. (2002). Collaborative filtering with privacy via factor analysis. In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '02, page 238–245, New York, NY, USA. Association for Computing Machinery,"https://doi.org/10.1145/564376.564419".

Cardosa, M., Wang, C., Nangia, A., Chandra, A., and Weissman, J. (2011). Exploring mapreduce efficiency with highly-distributed data. In Proceedings of the Second International Workshop on MapReduce and Its Applications, MapReduce '11, page 27–34, New York, NY, USA. Association for Computing Machinery,"https://doi.org/10.1145/1996092.1996100".

Chen, T., Zheng, Z., Lu, Q., Zhang, W., and Yu, Y. (2011). Feature-based factorization. CoRR, "http://arxiv.org/abs/1109.2271".

Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: Scalable online collaborative filtering. In Proceedings of the 16th International Conference on World Wide Web, WWW '07, page 271–280, New York, NY, USA. Association for Computing Machinery,"https://doi.org/10.1145/1242572.1242610".

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. Commun. ACM, ”https://doi.org/10.1145/1327452.1327492”, 51(1):107–113.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, 41(6):391–407.

Denning, P. J. (1982). Acm president’s letter: Electronic junk. Commun. ACM,”https://doi.org/10.1145/358453.358454”, 25(3):163–165.

Funk, S. (2006). Netflix update: Try this at home.

Giri, N. (2020a). 3 minute thesis, ”https://youtu.be/XUMc9QafDgw”.

Giri, N. (2020b). Recommendation system, ”https://github.com/NabinGiri/recommendation-engine”.

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. Commun. ACM,”https://doi.org/10.1145/138859.138867”, 35(12):61–70.

Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. information retrieval, 4(2):133–151.

Gong, S., Ye, H., and Dai, Y. (2009). Combining singular value decomposition and item-based recommender in collaborative filtering. In 2009 Second International Workshop on Knowledge Discovery and Data Mining, pages 769–772. IEEE.

GroupLensResearch (1998 (accessed March 22, 2020)). MovieLens Dataset,"https://grouplens.org/datasets/movielens".

Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In Proceedings of the 22nd annual international ACM SIGIR conference, pages 230–237, New York, NY, USA. ACM Press.

Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '95, page 194–201, USA. ACM Press/Addison-Wesley Publishing Co.,"https://doi.org/10.1145/223904.223929".

Hofmann, T. (2004). Latent semantic models for collaborative filtering. ACM Trans. Inf. Syst., 22(1):89–115.

International, M. and Media, O. (2004). Web 2.0 conference,"http://web.archive.org/web/20040602111547/http://web2con.com/".

Joyce, J. (2006). Pandora and the music genome project: Song structure analysis tools facilitate new music discovery. 23:14+40–41.

Kim, D. and Yum, B.-J. (2005). Collaborative filtering based on iterative principal component analysis. Expert Syst. Appl., 28(4):823–830,"https://doi.org/10.1016/j.eswa.2004.12.037".

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. Association for Computing Machinery, "https://doi.org/10.1145/245108.245126", 40(3):77–87.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, pages 426–434, New York, NY, USA. ACM,"http://doi.acm.org/10.1145/1401890.1401944".

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

Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing, ”http://dx.doi.org/10.1109/mic.2003.1167344”, 7(1):76–80.

Lisa Richwine, V. S. (2018). Netflix record subscriber growth dispels wall street worries, ”https://www.reuters.com/article/us-netflix-results/netflix-record-subscriber-growth-dispels-wall-street-worries-idUSKCN1MQ2R5”.

Luhn, H. P. (1958). A business intelligence system. IBM Journal of Research and Development, 2(4):314–319.

Netflix (2006). Netflix prize, ”https://www.netflixprize.com/”.

Netflix (2017). 2017 on netflix - a year in bingeing, ”https://media.netflix.com/en/press-releases/2017-on-netflix-a-year-in-bingeing/”.

Netflix Technology Blog, G. K. (2016). Selecting the best artwork for videos through a/b testing, ”https://netflixtechblog.com/selecting-the-best-artwork-for-videos-through-a-b-testing-f6155c4595f6”.

Owen, S. and Owen, S. (2012). Mahout in action.

Papadimitriou, S. and Sun, J. (2008). Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In 2008 Eighth IEEE International Conference on Data Mining, pages 512–521. IEEE.

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In Proceedings of KDD cup and workshop, volume 2007, pages 5–8.

Polat, H. and Du, W. (2005). Svd-based collaborative filtering with privacy. In Proceedings of the 2005 ACM symposium on Applied computing, pages 791–795.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In Proceedings of the 24th International Conference on Machine Learning, ICML '07, page 791–798, New York, NY, USA. Association for Computing Machinery,"https://doi.org/10.1145/1273496.1273596".

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In WWW '01: Proceedings of the 10th international conference on World Wide Web, pages 285–295, New York, NY, USA. ACM,"http://doi.acm.org/10.1145/371920.372071".

Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. T. (2000). Application of dimensionality reduction in recommender systems: A case study. In WebKDD Workshop at the ACM SIGKKD.

Satista (2019). Hours of video uploaded to youtube every minute as of may 2019, "https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/".

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth". In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '95, page 210–217, USA. ACM Press/Addison-Wesley Publishing Co., "https://doi.org/10.1145/223904.223931".

Statistics (2006). Pearson correlation, "https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/correlation-coefficient-formula/".

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the gravity recommendation system. Acm Sigkdd Explorations Newsletter, 9(2):80–83.

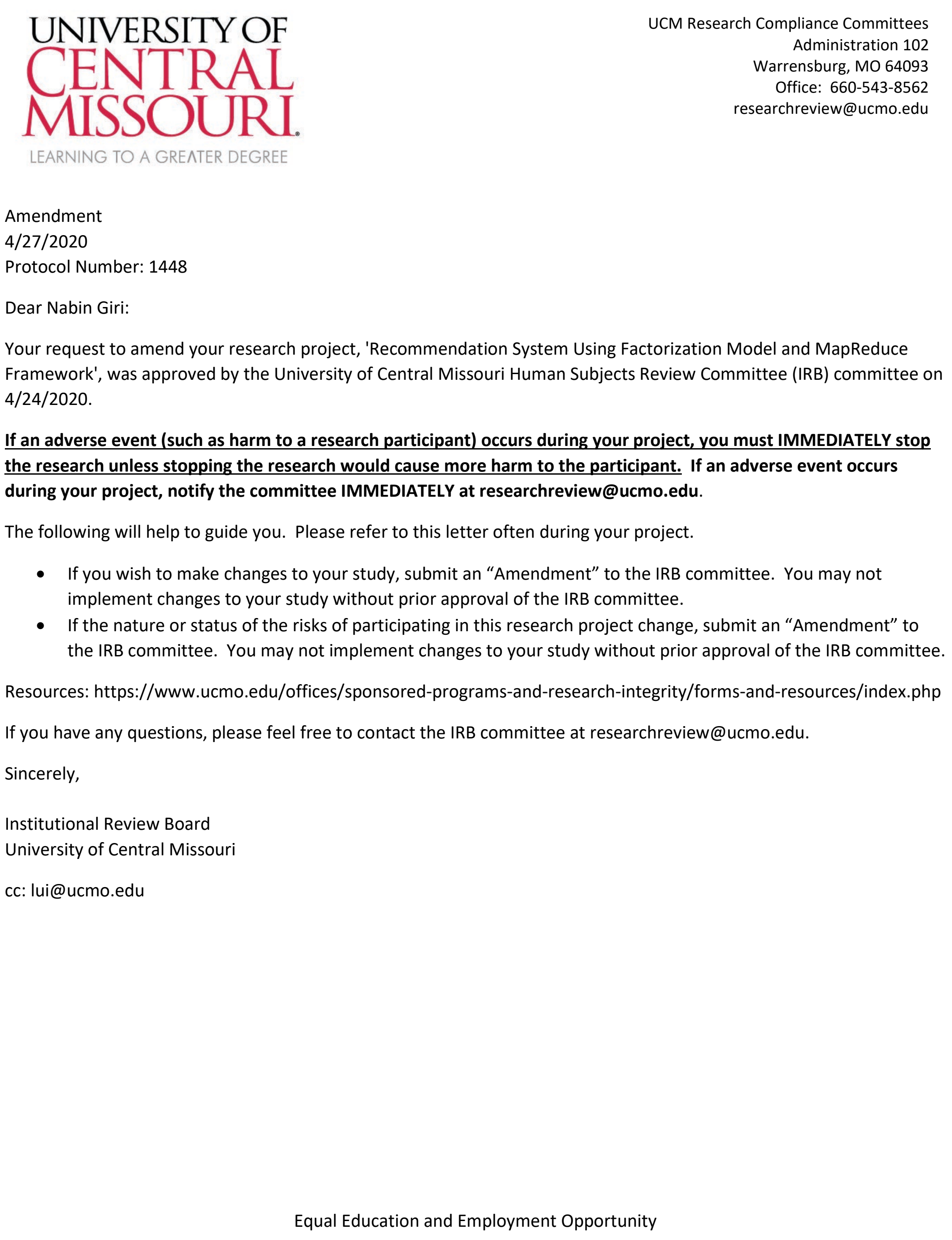
Vozalis, M. G. and Margaritis, K. G. (2007). Using svd and demographic data for the enhancement of generalized collaborative filtering. Information Sciences, 177(15):3017–3037.

White, T. (2009). Hadoop: The definitive guide , m. loukides, ed.

YouTube (2020). Youtube by the numbers,"https://www.youtube.com/about/press/".

Zheng, Z., Chen, T., Liu, N. N., Yang, Q., and Yu, Y. (2012). Rating prediction with informative ensemble of multi-resolution dynamic models. In Dror, G., Koren, Y., and Weimer, M., editors, KDD Cup, volume 18 of JMLR Proceedings, pages 75–97. JMLR.org,"http://proceedings.mlr.press/v18/zheng12a.html".

APPENDIX A
HUMAN SUBJECT APPROVAL



UCM Research Compliance Committees
Administration 102
Warrensburg, MO 64093
Office: 660-543-8562
researchreview@ucmo.edu

Amendment
4/27/2020
Protocol Number: 1448

Dear Nabin Giri:

Your request to amend your research project, 'Recommendation System Using Factorization Model and MapReduce Framework', was approved by the University of Central Missouri Human Subjects Review Committee (IRB) committee on 4/24/2020.

If an adverse event (such as harm to a research participant) occurs during your project, you must IMMEDIATELY stop the research unless stopping the research would cause more harm to the participant. If an adverse event occurs during your project, notify the committee IMMEDIATELY at researchreview@ucmo.edu.

The following will help to guide you. Please refer to this letter often during your project.

- If you wish to make changes to your study, submit an "Amendment" to the IRB committee. You may not implement changes to your study without prior approval of the IRB committee.
- If the nature or status of the risks of participating in this research project change, submit an "Amendment" to the IRB committee. You may not implement changes to your study without prior approval of the IRB committee.

Resources: https://www.ucmo.edu/offices/sponsored-programs-and-research-integrity/forms-and-resources/index.php

If you have any questions, please feel free to contact the IRB committee at researchreview@ucmo.edu.

Sincerely,

Institutional Review Board
University of Central Missouri

cc: lui@ucmo.edu

Equal Education and Employment Opportunity