Parallel Implementation of Basic Recommendation Algorithms

Theodosios Siomos

SID: 3308170022

Supervisors: **Konstantinos Diamantaras, Marios Gatzianas**

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

DECEMBER 2018

THESSALONIKI – GREECE

Abstract

A recommender system is a powerful tool that improves customer's experience through personalized recommendations. In order to provide recommendations, it analyzes the user behavior, such as the ratings available and browsing history. A well-designed recommender system is able to significantly increase the revenue of e-commerce web sites and applications.

There are many different models of recommender systems implemented. A successful recommender system must be accurate in its predictions and fast at the same time. A satisfied customer is very likely to be loyal to the web site or application. For that reason, the different models must be evaluated in terms of accuracy and performance.

The goal of this dissertation is to implement and parallelize three popular collaborative filtering methods. The methods implemented are the user-based collaborative filtering with the Pearson correlation, the item-based collaborative filtering with the adjusted cosine correlation and the model-based alternating least squares method.

These methods are also going to be evaluated for their accuracy and performance. For that reason, a set of different experimental metrics is used to evaluate their computing performance and their ability to provide accurate predictions.

Table of Contents

1 Introduction

The importance of Internet is growing rapidly and many tasks like electronic and business transactions are carried out online. Users have a tremendous list of options to choose from in a way that it can be challenging to select the most appropriate according to their preferences. There is an imperative need of recommender systems development in order to filter these information.

Recommender Systems or Recommendation Systems (RSs) are software tools and techniques whose main task is to provide recommendations on items to users. These recommendations refer to various decision-making processes such as what items to buy, which video to see, or which article to read [1]. In other words, recommender systems provide personalized recommendations, content and services to users [2]. Recommender Systems became an independent research area in the mid-1990s where the first papers on collaborative filtering started to appear. An important factor that recommender systems are used widely nowadays is the ease with which users can provide feedback about their preferences through the web sites [16]. A common way users can provide feedback is the by a rating system in which they assign numerical values to evaluate an item. The entity to which the recommendation is provided is often called user, and the product being recommended is also called item (book, song, movie, etc.). In other words, recommender systems help users to filter the content of a Web site by making them suggestions.

1.1 Goals of recommendation system

There are two distinct roles that recommender systems serve. The first one is from the perspective of the service provider. Merchants implement recommender systems on their Web sites in order to increase sales and therefore profit. The main operational and technical goals of recommender systems are [7]:

- *Increase the number of items sold:* Users are more likely to consume items that are close to their preferences.
- *Diversity:* Recommender Systems are considered to be useful when the recommended items have not been seen by the user in the past. Repeated recommended popular items can lead to reduction in sales diversity.

- *Increase user's satisfaction:* A well- structured Recommender System can increase the overall experience of the user with the Web site. The better the experience is, the longer users stay in the Web site.
- *Increase user's loyalty:* Users tend to be loyal in web sites that recognize them as old customers and treat them like valuable visitors.
- *Better understanding of user's needs:* The user's preferences that are either collected explicitly or predicted by the system, can be leveraged in the future in order to improve the management of the item's stock or adjust the advertising campaign properly.

From the perspective of the user, recommendations can improve the overall satisfaction with the web site. Through recommender systems, users can [23]:

- *Find items they prefer:* Users are more satisfied when an item that is recommended to them is close to their preferences.
- *Express their self:* For some users is crucial to contribute with their ratings and express their opinions. Recommender Systems can give this opportunity to them.
- *Help others:* Some users are happy to contribute to the community by providing their personal experience and expressing their beliefs.
- *Influence others:* There are some users whose main goal is to influence other people and direct them into purchasing specific items. This kind of actions can often be malicious in order to promote or disparage specific items.

A well-designed Recommender System must be able to satisfy both the roles and offer a helpful service.

1.2 Types of recommendations

The main two categories of recommender systems are the personalized and the non-personalized. In personalized recommendations, different users receive different suggestions according to their preferences such as favorite color and music genre. Personalized recommender systems are used by E-commerce web sites to suggest products to customers (based on their past activities). In non-personalized recommendations, the recommendations are independent on customers, so a group of users receive the same suggestions. These suggestions are based on what other customers said about the product on average. It is easy to understand that personalized suggestions are more

accurate and close to user's preferences. Non-personalized recommendations which are simpler to generate, often take place in magazines, television and radio [6]. They are considered to be automatic because it is not required user data and usually are short-lived.

1.3 The Recommendation Problem

The main target of Recommender Systems is to provide suggestions about new items or to predict the utility of a specific item for a particular user. The recommendation problem can be modeled in two different ways:

- *Prediction version of the problem:* The prediction is expressed as a numerical value. For m users and n items in the system we have an incomplete m x n matrix where the observed values are used for train. The unobserved (missing) values are the value to be predicted by the system. That is the reason, this problem is called as the matrix completion problem because we are trying to fill the missing values from the m x n matrix. The predicted values should be in the same numerical scale as the observed values [7][9].
- *Ranking version of the problem:* The recommendations are expressed as a list of N items with $N \leq n$ which the user is expected to like the most. Usually merchants are more interested to provide the top-k items for a target user or the top-k users for a target item rather than predicting all the ratings from the m x n matrix. This problem is also referred as the top-k recommendation problem [7][9].

1.4 Implicit and Explicit data

The information that is used by recommender systems is user's feedback and can be divided in two categories: Explicit and Implicit data. Explicit data are the numerical ratings users provide in order to evaluate items. The ratings are usually on a scale which indicates how much the item is liked or disliked. Most of the times, ratings are interval-based, where a discrete set of ordered numbers is used to quantify like or dislike. There is a variation between web sites on the scale of possible ratings. The most common models are the 5-star (Amazon) and 10-star (IMDb). Another type of system is the binary system that explicitly indicates whether a user likes an item or not and nothing else (YouTube). A special case of ratings is the unary ratings where a user can specify a liking for an item

but there is no option to dislike an item (Facebook) [5]. Unary ratings are widely common especially in implicit data. On the other hand, implicit data refer to user's actions and not their explicitly specified ratings. Examples of implicit data are the duration of browsing, click times, movements of mouse, refresh of web site, bookmarks and download of Web content [3]. Both the explicit and implicit data are easily obtainable data. However the implicit data are more difficult to be analyzed. Nevertheless, implicit data provide more information about the user otherwise might not be provided. The earlier recommender systems were trying to solve an explicit-data prediction problem while the modern ones use a set of different techniques to accomplish their task [4].

2 Categories of Recommender Systems

Recommender Systems can be categorized according to the information they use to recommend items into [6]:

- *Collaborative Filtering Systems:* Uses information from people with similar tastes and preferences to recommend items to the user.
- *Content-based Filtering Systems:* Uses information from items the user preferred in the past.
- *Demographic Filtering Systems:* Uses demographic information such as age, gender, etc. to identify types of user.
- *Hybrid Recommender Systems:* These systems combine collaborative and content-based methods.

2.1 Collaborative Filtering Systems

These systems analyze existing active customers with similar preferences and characteristics of the current customer to build recommendations [3]. The basic idea is that the missing ratings can be predicted because the observed ratings are usually high correlated between various users and items. For example, two users of the system have similar preferences according to their observed ratings. It is very possible that the ratings in which only one of them has specified a value, are also possible to be similar. Most of the models for collaborative filtering try to exploit either inter-item correlations or inter-user correlations to predict the missing values [5]. Collaborative filtering is the most successful technology used in recommender systems and it is used widely on the internet [8]. There are two main categories that collaborative filtering algorithms can be grouped: memory-based (or heuristic based) and model-based [8].

2.1.1 Memory-based

This type of methods were the earliest collaborative filtering algorithms. The user-item ratings are stored in the memory and they used directly to predict the missing values. Once some statistical techniques are applied to form neighborhoods with similar users, the system is ready to predict the unobserved values. That is the reason memory-based systems are often called neighborhood-based

systems. Neighborhoods can be formed in two different approaches: The user-based and the item-based approach.

- *User-based collaborative filtering:* The predicted ratings of a target user are provided by users with similar preferences. The main idea is to determine users with similar tastes to the target user A. The k-closest users to A are used to make predictions. In order to find the k-closest users, similarity functions are computed between the target user and all the users of the system.
- *Item-based collaborative filtering:* In this case, the predicted ratings for a target item B by user A are provided by similar items to target item B the user A has rated. Here, the main idea is to find a set with the k-closest items to target item B to predict the missing values. In order to find the k-closest items, similarity functions are computed between the target item and all the items of the system.

The advantages of memory-based systems are [1]:

- *Simplicity:* They are simple to implement.
- *Justifiability:* The predicted missing values can be justified and explained comprehensibly.
- *Efficiency:* They do not require constant training in terms of large commercial applications.
- *Stability:* There is a little influence when a new user or item is added to the system.

The main disadvantage of this model is that memory-based algorithms do not work very well with matrices that are sparse. For example, there is a chance that the k-closest users have not seen all the movies as a total. The predicted ratings for that movies lack full coverage.

2.1.2 Model-based Methods

In this type of methods, the system imitates the interactions of user-item with factors which represent the hidden characteristics of the user and the items in the system. The model is trained with the observed ratings (train dataset) and the trained model is used to predict the missing values. Some examples of model-based methods are the decision trees, Bayesian methods, latent semantic analysis and support vector machines.

The main advantage of model-based methods compared to memory-based is the improved prediction performance. The combination of the aforementioned methods provide the most accurate results [5].

2.1.3 Collaborative Filtering Challenges

Collaborative filtering is constantly evolving and has been very successful in both the research and practice area. Nonetheless, there is space for further improvement. The main challenges collaborative filtering recommender systems face are [8]:

- *Scalability:* The first challenge refers to the scalability of collaborative filter algorithms. These algorithms often search thousands or even millions potential neighbors in real time. Many of the existing algorithms have performance issues with users who have big amount of information. This situation can result to further reducing scalability due to the reducing number of neighbors can be searched per unit of time. It is needed that these systems can provide recommendations in real time as fast as possible.
- *Quality of recommendations:* The second challenge refers to the improvement of quality in recommendations. Consumers are more satisfied when the recommendations are accurate and interesting. The more satisfied the user is, the more loyal tends to be to the Web site. If a user purchases an item and finds out that he does not like the item, it is less possible that he will trust the recommender system again.

These two challenges are in a constant conflict. Recommender systems need to be both accurate and fast in order to provide a satisfying experience to the user. For this reason, it is vital to solve these challenges at the same time.

The disadvantages of collaborative filtering technique are:

- *Cold Start Problem:* The cold start problem refers to the new users or items added to the system. When a new entity is added, there is no information related to them. For example, when a new item occurs there are no previous ratings about the item and therefore it cannot be recommended [7].
- *Scalability:* As the system grows, there is an increasing need of resources in order to give accurate and fast predictions. A system with millions of users and items requires a significantly amount of computational power [1].
- *Sparsity:* In recommender systems, the number of observed ratings is tiny compared to the number of missing values to be predicted. The success of collaborative filtering lies on the availability of a critical mass of users and items [2]. Sparsity is the lack of critical knowledge that affects collaborative filtering.

2.2 Content-based Filtering Systems

Content-based filtering, also called cognitive filtering, recommends items to users based on a comparison between the content of the items and the user profile [1]. In these systems, the descriptions of the items, which are labeled with ratings are used as the training set in order to create a user-specific classification or regression problem. The descriptions of the items and the profiles of the users play an important role for content-based filtering. The recommendations are based on the similarity count.

The advantages of content-based systems are [1]:

- *Independence:* The system can provide exclusive ratings which are used by the user to build their profile.
- *Transparency:* The system is able to justify the recommendations to the user.
- *Effective prediction for new items:* The system is able to recommend items that they have not been rated yet.

The disadvantages of content-based systems are [5]:

- *Obvious recommendations:* There are many cases of obvious recommendations due to the use of keywords or content.
- *Not effective prediction for new users:* The system need the history of user's ratings in order to provide recommendations

2.3 Demographic Filtering Systems

This kind of system uses pre-existing knowledge of demographic information such as age, gender and nationality for the recommended items as the basis for recommendations [6]. We could say, that demographic systems are stereotypical because they are based on the assumption that all the users who belong in the same demographic group have similar tastes and preferences. The main advantage of these systems is the lack of need in requiring user's history in order to make predictions. On the other hand the lack of such data may often lead to inaccurate predictions.

2.4 Hybrid Recommender Systems

It is another more sophisticated category of recommender Systems. It combines different approaches in order to overcome their limitations. The most popular one is the mix of collaborative and content-based filtering. There are many ways to combine these methods in order to solve their problems. Examples of them are [4]:

- Implementation of both filtering separately and combine the results.
- Embody the characteristics of content-based filtering to collaborative filtering adding characteristics from the first to the latter and vice versa.

Hybrid Systems achieve the best results in predicting the missing values of the system.

3 Evaluation of Recommender Systems

There are many ways to evaluate the performance of the various recommender systems. Systems that generate predictions as numerical values should be considered separately from those which have the form of top-k recommendations [9]. The evaluation of collaborative filtering has many things in common with that of classification or regression. The reason behind this is that collaborative filtering can be seen as a generalization of the classification or regression problem [7]. Evaluation systems must be well-designed in order to measure the actual effectiveness of Recommender Systems. Most of the times, evaluation is divided in two categories. The first category is referring to how accurate the results are. The second one evaluates the performance of recommender systems in terms of time and space requirements [7]:

3.1 Metrics evaluating Prediction Quality

Recommender Systems provide predictions as numerical values. These predictions should reflect the preferences and tastes of the user. The two most common ways to evaluate those predictions are in terms of accuracy and coverage [2][9].

3.1.1 Accuracy

One of the most important measures with which recommender systems can be evaluated is accuracy. There are two methods in order to measure accuracy. The first one is based on the statistical accuracy and the second one on decision support accuracy.

Methodology of accuracy evaluation

Let R be the ratings matrix in which r_{uj} refers to the observed rating (true rating) by user i for item j. For the purpose of evaluation, the observed ratings from R are divided in two different sets, the training and test set. An observed rating cannot be in both of the sets because there is a risk of overfitting. Let S be the total number of observed ratings in R and T a small subset of them. Then, the training set consists of S – T ratings. The test set is the T subset used for evaluation. Recommendations systems provide estimations for the true values as $\hat{r}_{ij}$. The entry specific error is given by $e_{ij} = \hat{r}_{ij} - r_{ij}$. This error can be used in different ways in order to measure of recommender Systems. The most common statistical accuracy metrics are:

Mean absolute error (MAE)

Calculation of MAE involves summing the magnitudes (absolute values) of the entry-specific errors and dividing them with T.

$$MAE = \frac{\sum_{(i,j)\in T}\left|e_{ij}\right|}{|T|}$$

Mean squared error (MSE)

Calculation of MSE involves summing the values of the squared entry-specific errors and dividing them with T.

$$MSE = \frac{\sum_{(i,j)\in T} e_{ij}^2}{|T|}$$

Root mean squared error (RMSE)

RMSE is the root of the MSE metric and is given by:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{\sum_{(i,j)\in T} e_{ij}^2}{|T|}}$$

Assessment of MAE and RMSE

MAE and RMSE are the most famous metrics in evaluating the accuracy of recommender systems both in the research and commercial area. While they have been used for many years, there is no clear answer on which metric is the best. The main difference between them is that MAE gives equal weight to all errors, while RMSE penalizes variance because it gives error with large absolute values more weight than errors with small absolute values [10]. The sensitivity of the RMSE to outliers is the most common concern with the use of this metric. A few bad predictions can ruin the RMSE measure. On the other hand, the biggest advantage of RMSE compared to MAE is the lack of using the absolute value, which is highly undesirable in many mathematical calculations [10]. Supporters of MAE, claim that this metric is the most natural measure of average error magnitude and suggest that recommender systems accuracy to be based on it [11].

3.1.2 Coverage

Even though a recommender system can be very accurate, it may not be able to recommend a specific proportion of items, or it may be incapable of recommending ever certain items to users. This inability to generate predictions for all the items and users is due to the high level of sparsity recommender systems face [7]. For extreme levels of sparsity, it is possible that not even one prediction can be generated by the system. A low level of coverage indicates that the system is not able to assist the user effectively in finding interesting items that he has not rated yet. On the other hand, a high level of coverage indicates that the system is capable of providing recommendations that the user is expected to enjoy more [9]. Let n_i be the items that a user u has rated and np_i be the items for which the Recommender System was capable of providing predictions where $np_i \leq n_i$, then the coverage can be given by:

$$coverage = \frac{\sum_{i=1}^{m} np_i}{\sum_{i=1}^{m} n_i}$$

The total coverage is defined as the fraction of items for which the system was able to generate a prediction over the total number of items that all available users rated in the initial R ratings matrix [9].

3.2 Metrics evaluating performance

Apart from evaluating recommender systems in terms of accuracy, there are performance metrics related to time and storage requirements:

- *Response Time:* It is a common performance metric which is implemented for various reasons in different technology areas. In the area of recommendation systems, it measures the time elapsed between the moment a user requested a recommendation and the moment of response by the system. The request can be either in form of top-k recommendations or a prediction. Response time is a very valuable metric because users tend to be more satisfied with quicker results.
- *Storage requirement:* It is quietly common that recommender systems provide predictions from a dataset which has millions or even billion users and items. It is very helpful to evaluate how the system leverage the resources provided to them. For that reason, storage requirement can be categorized in main memory requirement and secondary storage

requirement. The first one refers to the online space usage of the system while the latter one to the offline.

- *Training time:* Most of the recommender systems require a training phase in order to provide results. For example, a matrix factorization system needs to calculate first the latent factors. Usually, the training part is computed offline. Nonetheless, the training time must be an acceptable number in order that the system to be able to make recommendations.

3.2.1 Parallel computing and Recommender Systems

Parallel computing is a type of computing where many calculations or the execution of processes are carried out simultaneously [15]. Recommender Systems address to a problem with a quite big dataset. The total number of users and items can be millions or even billions. It is crucial that the independent calculations of the recommender systems to be parallelized in order to utilize systems with multi-core processors and GPUs. Utilizing all the resources available can reduce significantly both the training and online time recommender systems need to give predictions. For that reason, there is a need to be metrics to evaluate how problem sizes and system sizes influence the performance of parallel computers and parallel algorithms [14]. The most common methods to measure the performance of parallel systems are:

Speedup

This measure is used to quantify the performance gain from a parallel computation of a fixed-size application over its computation on a single core [15]. It is given by:

$$S(p) = \frac{T(1)}{T(p)}$$

where T(1) is the time of execution with one processor and T(p) is the execution time with p processors.

Efficiency

It measures the time percentage for which a machine is usefully employed in parallel computing. It is defined as the ratio of speedup to the number of processors [15].

$$E(p) = \frac{S(p)}{p} = \frac{T(1)}{p\ x\ T(p)}$$

Scalability

This measure evaluate the ability of a parallel machine to improve performance as there are increases in the size of the application problem and in the size of the system involved [15].

Parallel computing with GPUs

In the past years, there is a rapid development of graphics processing units (GPUs). GPUs can handle highly parallel workloads and execute thousands of concurrent threads [16]. By the introduction of CUDA (Compute Unified Device Architecture), GPU computing has become famous to the point that it is considered to be state of the art in many applications. CUDA is a parallel computing platform and application programming interface (API) created by Nvidia. Nowadays is used in many application related to big data. The CUDA API can be leveraged by third party frameworks like TensorFlow and PyTorch. In the recommender systems area, GPUs can significantly speed up the execution time of both offline and online phase. However the amount of memory capacity and control logic which are available on a GPU are limited. As a result, there can be a number of runs required in order to completely process a big dataset [16][17]. However, GPUs are very important in the area of recommender systems and are used widely especially in the last decade [16][17][18].

4 Collaborative Filtering

4.1 Neighborhood-Based Collaborative Filtering

Neighborhood-based collaborative filtering is among the most famous methods for recommender systems. They are also referred as memory-based algorithms. This technique belongs to the earliest implementations for collaborative filtering. It analyzes the relations between users (neighbors) and mutual-dependencies between items in order to identify new user-item associations [12]. They can be viewed as generalizations of k-nearest neighbor classifiers, which are used widely in the machine learning area. There are two main types of neighborhood-based algorithms [7]:

- *User-based collaborative filtering:* In this method, the ratings from users with common preferences and tastes are leveraged in order to make predictions for a target user A. The predicted ratings for user A are calculated as the weighted average values of the neighborhood that was formed. For example, two users A and B have very similar ratings on items that they both rated. User-based collaborative filtering suggests that with the observed ratings from user B, someone can predict the unobserved values from user A and vice versa.
- *Item-based collaborative filtering:* Let C be the target item to make recommendations. The first step is to find the most similar items to C and form a neighborhood. In order to make a prediction for any user for the target item C, we use the neighborhood that we formed and use the observed ratings of the user. The weighted average of the ratings in the neighborhood is used to predict the ratings of any user for the target item C. For example, two items C and D are very similar and they belong to the same neighborhood. A user has a positive rating about the item C and has no rating about the item D. Item-based collaborative filtering suggests that the user is very possible to like the item D.

The main difference between the user-based and item-based collaborative filtering is that the missing ratings in the first method are predicted by forming neighborhoods of users and leveraging their ratings, whereas in the second method are predicted by using user's observed ratings and forming neighborhoods on items. In the first method, neighborhoods are formed by finding the similarities among users (rows of rating matrix). In the second method, neighborhoods are formed

by calculating the similarities between the items (columns of rating matrix). The two methods share a complementary relationship [7].

4.1.1 User-based neighborhood models

Let R be the ratings matrix of size m x n where m are the number of users and n the number of items. The set of observed ratings from user u is denoted by I_u. In the same way, the set of observed ratings from user v is denoted by I_v. The set of items that both users rated is $I_u \cap I_v$. It is quite common that this set to be empty because the rating matrices are very sparse. The set $I_u \cap I_v$ is used to calculate the similarity between the users. In order to form the neighborhood of user u, all the similarities between the user u and the users of the system are calculated [7].

4.1.2 Item-based neighborhood models

In the item-based approach, the neighborhoods are formed in terms of items rather than users. This method examines the set of items the target user has rated and computes how similar they are to the target item i. Then, it selects the k-closest items and at the same time their corresponding similarities. When the k-closest items are found, the system is able to predict the rating of the target item by computing the weighted average of the target's user ratings on these similar items [19]. The main process of this method is to find the similarities between all the items of the system. The first step in computing the similarity between items i and j is to find the mutual set of users who have rated these to items. Defining that set allows to find the similarity between those items.

Similarity coefficients

There are many statistical coefficients that can be used as similarity measures for the neighborhood-based models. The greatest challenge in this kind of systems is to form neighborhoods containing as much similar users or items is possible [12]. The most famous coefficients in both the literature and e-commerce are the Pearson correlation and the cosine similarity. Both of them can be used either in user-based or item-based neighborhood models. However, Pearson correlation provides better results when it is used in the user-based models while cosine similarity in item-based neighborhood models.

4.1.3 Pearson correlation

The Pearson correlation is the most common measure for neighborhood-based collaborative filtering and it is used by many recommender systems. The coefficient is computed on the set $I_u \cap I_v$

which contains the mutual rated items of user u and v. It is computed between all the users of the system (rows of ratings matrix) and is given by:

$$Sim(u,v) = Pearson(u,v) = \frac{\sum_{k \in \mathrm{Iu} \cap \mathrm{Iv}} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in \mathrm{Iu} \cap \mathrm{Iv}} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in \mathrm{Iu} \cap \mathrm{Iv}} (r_{vk} - \mu_v)^2}}$$

where

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|}$$

The computation of values of μ_u and μ_v can be in different ways which lead to different results. With the first way which is simpler and computationally less costly, the mean is calculated for all the user's observed ratings on items. In the second approach, the mean is computed only with the ratings of the set which contains the mutual rated items. There is not a clear answer on which method provides better results [7].

The next step after the Pearson correlation is computed between the target user and all the other users is to form target user's neighborhood. The neighborhood is consisted of the top-k highest correlated users. The weighted average of neighborhood ratings are used to predict the ratings of the unobserved items of the target user. A serious problem that may occur is that users provide ratings having a different sense of rating items. For example, some users tend to rate all the items highly and some others low. In order to fix this problem, the ratings must be mean-centered in row-wise fashion, before determining the average rating of the neighborhood. The mean-centered rating s_{uj} is given by:

$$s_{uj} = r_{uj} - \mu_u, \quad \forall u \in \{1 \dots m\}$$

Let $P_u(j)$ be the set of k-closest users to the target user u. In this set may exist users with very low correlations even negative. These users can be filtered out to get better results in predicting the missing values. The prediction function which is the same for all neighborhood-based systems is:

$$\widehat{r_{uj}} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |Sim(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Sim(u,v)|}$$

The Pearson correlation coefficient is a very successful method to measure the similarities between users or items. However, there are some disadvantages which limit its effectiveness [12]:

- When the Pearson correlation is computed between two users, only the set with the mutual observed items is taken into consideration. Although the users may have rated many movies, it is very possible that the mutual set of rated movies to be small. For that reason, Pearson correlation is not regarded as a trustworthy measure of similarity.
- The similarity between two users can be computed only in the case of having mutual rated items. If the users do not have mutual rated items, Pearson correlation is unable to define a similarity between them even if they are indeed high correlated.
- When the data of the system are too sparse, Pearson correlation struggles in giving accurate predictions. Most of the times, the ratings matrix R of a recommendation system is very sparse.

4.1.4 Raw cosine similarity

The main difference between the similarity computation of user-based and item-based neighborhood methods is that in the first case, the similarity is computed between the users (rows) while in the latter case between the items (columns). The coefficient is computed on the set $I_u \cap I_v$ which contains the mutual users that rated both the items u and v.

$$Sim(u,v) = Raw\ Cosine(u,v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r^2{}_{uk}} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r^2{}_{vk}}}$$

Computing the similarity between two items using the raw cosine similarity has a serious disadvantage. It does not take into consideration the different rating scales users may have. For that reason, the Pearson correlation is more preferable due to the bias adjustment effect of mean-centering [7]. The prediction function which is the same for all neighborhood-based systems is:

$$\widehat{r_{uJ}} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |Sim(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} Sim(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |Sim(u,v)|}$$

4.1.5 Adjusted Cosine Similarity

In order to solve the problem of users' rating biases the adjusted cosine similarity is introduced. The solution of this method is subtracting the corresponding user average from each r_{uj} observed rating so a mean-centered matrix is created [19]. The adjusted cosine similarity between the items (columns) i and j is given by:

$$Sim(i,j) = Adjusted\ Cosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s^2{}_{ui}} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s^2{}_{uj}}}$$

The coefficient is computed on the set $U_i \cap U_j$ which contains the mutual users that rated both the items i and j. The Pearson correlation can be also used in order to find the correlations between the items (columns) in the item-based neighborhood methods. However, the adjusted cosine provides better results in most of the times [7].

Let t be the target item for user u. In order to predict the missing rating r_{ut}, we first find the top-k closest items to the target item t by computing the adjusting cosine similarity. Let the top-k closest items to item t, for which the user u has observed ratings, be denoted by $Q_t(u)$. The weighted average of these ratings is the predicted value and it is given by [7]:

$$\widehat{r_{ut}} = \frac{\sum_{j \in Q_t(u)} Adjusted\ Cosine(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |Adjusted\ Cosine(j,t)|}$$

The basic idea behind this method is to take advantage of user's own ratings on similar items in order to make the prediction.

Strengths and limitations of neighborhood-based techniques

The advantages of neighborhood-based methods are closely related to their simplicity and intuitive approach [7]. For that reason, they are easy to be implemented and debugged. Furthermore, it is easy to interpret the predictions and give justifications to the users about the recommendations. On the other hand, in model-based systems this kind of justifications are not easy to be given. One more advantage is that recommendations are quite stable despite the addition of new users or items in the system. All the aforementioned strengths are the reason neighborhood-based systems to be very popular until today and used in many recommender systems.

The basic limitation of neighborhood-based techniques is the computation time needed for the offline phase. The biggest E-commerce web sites operate at a scale that stress the direct implementation of these systems [19]. The offline computation requires at least $O(m^2)$ time in the user-based approach and $O(n^2)$ in the item-based approach where m and n are the total users and items of the system. Another crucial limitation is the difficulty in presenting accurate predictions when the ratings matrix is too sparse [12].

Comparing user-based and item-based methods

Usually, the item-based approach gives better results comparing to the user-based approach. The reason behind this is that item-based methods leverage user's own ratings in order to make a recommendation. This does not apply for user-based methods in which the ratings are taken from users who may have things in common but different tastes [7]. The outcome is that item-based methods are more accurate than user-based.

On the other hand, user-based methods tend to be more diverse in comparison to item-based methods. Diversity is the ability of a recommender system to provide recommendations in a way that they are unique between them. A recommender system which makes similar and obvious recommendations is not very effective. There are situations where item-based methods recommend items that are not fresh to the user according to his purchase history [7].

Item-based methods are considered to be more stable with changes to the ratings. This happens for two basic reasons. The first reason is that the number of users in the system is most of the times larger than the items. Two users may have a small set of mutual rated items but two items are more possible to have a larger set mutual users that have rated them. As a result, user-based systems with the addition of few ratings can change drastically the similarity values [7]. The second reason is that in large E-commerce systems, new users are added more frequently than items. In this situation the finding of item neighborhoods is stable while the user neighborhoods are constantly changing. The outcome of this situation is that user neighborhoods need to be updated more regularly than item neighborhoods and that is more costly.

Performance implications and efficient implementation

Modern recommender systems have millions of active users and items to recommend. They have to be as accurate as possible and fast at the same time. These systems can be stretched by neighborhood-based systems to the limits. Especially in the user–based method, computation can be a performance bottleneck which can make the whole process unsuitable for real-time recommendations [19].

The first step of neighborhood-based algorithms is to compute the similarities between all the entities of the system. It is advisable, that the whole procedure to be divided in two phases. The offline phase where all the heavy computation is made and the online which gives the final prediction to the user [7]. In the offline phase, the user-user or item-item similarities are computed

and the user or item neighborhoods are formed. In the online phase, neighborhoods are leveraged in order that the recommender system give predictions. In the case of user-based systems, the formation of the neighborhood of a target user is computed in O(m·n') time where n' is the is the number of the observed ratings of the user. The overall computation of neighborhoods requires $O(m^2 \cdot n')$ time. The overall computation time for the item-based method is $O(n^2 \cdot m')$ respectively where m' is the number of observed ratings of an item.

In order to compute the online phase, recommender systems need to store all the pairs of similarities between users or items. The space requirements of user-based methods are $O(m^2)$ while the space requirements for item-based methods are $O(n^2)$. Most of the times, the number of users in the system exceed the number of the items. For that reason, the space requirements of user-based methods are greater than the item-based.

The online computation of a predicted value is O(k) for both the user-based and item-based method, where k is the size of the user or item neighborhood. The overall computation for all the items of the system is O(k·n) while for all the items is O(k·m). The aforementioned computation is the same for both the user-based and item-based methods.

The conclusion is that the heavy computation of neighborhood-based algorithms is handled in the offline phase. The offline phase needs to be executed as many times as possible in order that the system to be updated. The online phase is very efficient and can give fast results. This is not a very big problem for the big E-commerce web sites because the heavy tasks are computed without users noticing any bottleneck when recommendations are given to them.

4.2 Model-based collaborative filtering

In model-based systems, a summarized model of the data is created using supervised and unsupervised machine learning methods. Examples of these methods are Bayes classifiers, decision trees regression models and support vector machines. These methods can be used in the collaborative filtering problem because the matrix completion problem is a general case of the traditional classification and regression problem [7]. However, there are obstacles in order to generalize the classification or regression models. The most important one is the sparsity of the ratings matrix. The main advantages of model-based recommender systems over the neighborhood-based are:

- *Training speed and prediction speed:* The offline phase of neighborhood-based systems can be very slow because the computation requires $O(m^2 \cdot n')$ for user-based and $O(n^2 \cdot m')$ for item-based models. On the other hand, model-based are most of the times faster than neighborhood-based systems in the training speed of the model. Once the model is built, the prediction time is almost instant making them very efficient [22].
- *Avoiding overfitting:* Overfitting is a very important issue in the field of machine learning. Overfitting occurs when a model learns every detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. Model-based methods are able to use techniques like regularization in order to cope with overfitting making them robust [7].
- *Space efficiency:* The trained model of neighborhood-based methods have complexity of $O(m^2)$ for user-based and $O(n^2)$ for item-based methods. The trained model of model-based systems are much more efficient.
- *Allows the usage of implicit feedback:* One basic strength of model-based systems is that allow incorporation of additional information [20]. The structure of modern E-commerce web sites provides plenty implicit feedback from users such as search patterns, mouse movements and purchase history. In many cases, explicit data are completely unavailable. Model-based can leverage these data and give more accurate predictions than neighborhood-based models. They can even completely based on implicit data without using explicit data at all.

In general neighborhood-based systems are simpler than model-based systems and easier to be implemented. They were the first systems to be implemented in the field of recommender systems. However, model-based systems are in many situations better in terms of accuracy and efficiency. For that reason, modern E-commerce web sites implement model-based techniques in their Recommender Systems.

4.2.1 Latent Factor Models

Latent factor models are an alternative approach to Collaborative Filtering and are able to uncover latent features that can explain observed ratings [22]. The rows and columns of ratings matrices are high correlated. Latent Factor Models leverage this fact in order to build redundancies. These redundancies can be used to compute a low-rank matrix that approximates the resulting ratings matrix. The fully specified low-rank approximation can provide very accurate predictions on the

missing values of the ratings matrix. They have repeatedly better accuracy than other methods such as neighborhood models and restricted Boltzmann machines [21]. Latent Factor Models are the most used techniques in today's Recommender Systems and they are considered to be state of the art.

Problem Formulation

A low-rank latent factor model associates with each user and with each item a vector of rank n_f, for which the component measures the tendency of the user or item towards a certain factor or feature [21]. For example, a latent feature can represent a genre in the area of music (e.g. Jazz, Pop). The effectiveness of latent factor models comes with the factorization of the initial matrix. Factorization is a generalization of approximating a matrix when dimensionality reduction can be occurred because there are high correlations between rows and columns. Most of the dimensionality reduction methods can be expressed as matrix factorizations.

Let R be the ratings matrix of mxn size and rank $k \ll \{m, n\}$, where m is the total number of users and n is the total number of items. In the case of all the ratings of the matrix R are observed, R can be expressed in the following product form of rank-k factors:

$$R = UV^T$$

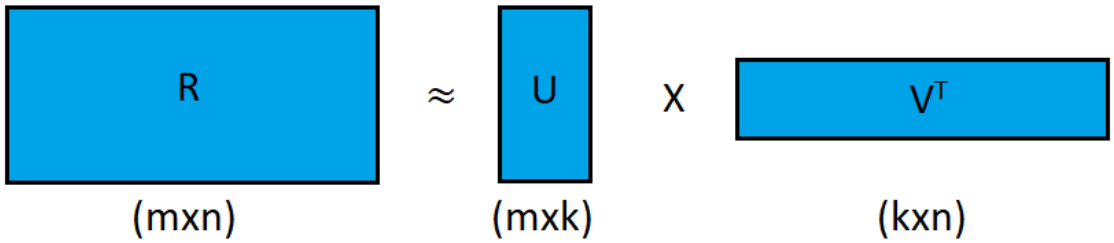
U is a matrix of $m \times k$ size and V is a matrix of $n \times k$ size. The set of all linear combinations of the rows of a matrix is referred as the row space of the matrix. Respectively, the set of all linear combinations of the columns of a matrix is referred as the column space of the matrix. The rank of both the row and column space of R is k [7]. A low-rank matrix factorization procedure takes as inputs the numerical ratings of R and determines the low-rank user U and item V where the columns in these matrices represent a latent factor for a single user or item respectively [21].

Linear algebra enables the factorization of any rank-k matrix. The number of the possible factorizations is infinite. The latent factors model is closely related to the singular value decomposition (SVD) problem with the main difference that the SVD of a matrix with missing values is undefined. [21]. Even if the matrix R has larger rank than k, it can be approximated as:

$$R \approx UV^T$$

For low values of the rank, it is still possible to approximate R. The reason behind this is that there is not a need to have many observed ratings to estimate the latent factors. This is very crucial because most of the times the data of the Recommender Systems are extremely sparse. The error of this

approximation is given by $\|R - UV^T\|^2$ where $\|\cdot\|^2$ represents the sum of squares of the entries in the resulting residual matrix (R-UV^T). This is also called as the (squared) Frobenius norm of the residual matrix [7].



4.2.1.1The matrix factorization problem

The columns of U or (V) are called latent vectors or latent components, while the rows of U (or V) are called latent factors. A rating r_{ij} can be approximated as the product of the i_{th} user and j_{th} item factor as:

$$r_{ij} \approx \overline{u_i} \cdot \overline{v_j}$$

The computation of U and V matrices is the first and most important step of this kind of systems. Once they are built the Recommender System is able to provide predictions. For that reason, it is crucial that the factorization of R to be quick.

4.2.2 Unconstrained Matrix Factorization

The basic form of matric factorization is the unconstrained one where no constraints are imposed on the factor matrices U and V. The goal of matrix factorization is to form the two matrices as accurate as possible in order that the specified matrix R to be as close it can be to UV^T. This is an optimization problem with respect to the matrices U and V and has the following form:

$$Minimize\ J = \frac{1}{2}\|R - UV^T\|^2$$

In this formula, $\|\cdot\|^2$ denotes the squared Frobenius norm of the matrix that equals to the sum of squares of the matrix entries. So, the objective function is equal to the sum of squares of the entries of the residual matrix$(R - UV^T)$. The objective function need to be as small as possible in order that R to be approximated better. It is a quadratic loss function which quantifies the loss of accuracy in estimating the matrix R with the use of low-rank factorization [7]. The above formula is defined where the matrix R is full specified by observed ratings. In real problems this is not the case. The

dataset may be extreme sparse. For that reason the formula need to be redefined in terms of the observed ratings. Once the latent factors are built, the entire ratings matrix can be approximated as UV^T in one shot [7].

Let all the observed ratings of R be denoted by S. Let $i \in \{1 \dots m\}$ and $j \in \{1 \dots n\}$ where i is the index of a user and j is the index of an item. The set S of observed ratings is defines as:

$$S = \{(i,j):\ r_{ij}\ is\ observed$$

The matrix R is not fully specified because it has missing values. If R can be factorized in two fully specified matrices U and V then all the values of R including the missing ones can be approximated as:

$$\widehat{r_{ij}} = \sum_{s=1}^{k} u_{is} \cdot v_{js}$$

The difference between an observed and a predicted is given by $e_{ij} = \left(r_{ij} - \widehat{r_{ij}}\right) = \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)$.

$$Minimize\ J = \frac{1}{2} \sum_{(i,j) \in S} e_{ij}^2 = \frac{1}{2} \sum_{(i,j) \in S} \left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js} \right)^2$$

The above objective function computes the error only of the observed ratings in S.

4.2.3 Regularization

In real world applications, the ratings matrix R is very sparse. This fact can cause many problems in predicting its missing values in many ways. If the observed ratings are very limited there is a great chance overfitting to be occurred. Overfitting is the problem that prevents the model to have a good generalization. In the first implementations, systems used agents to impute the values of the missing ratings and make the sparse matrix dense. However, this method can be very computationally expensive and it increases the amount of data [20].In order to face this problem many modern models use the regularization technique. Regularization reduces the tendency of the model to overfit by inserting a bias into it [7].

The basic idea is to discourage very large values of the coefficients in U and V in order to stabilize them. For that reason, a regularization term $\frac{\lambda}{2}(\|U\|^2 + \|V\|^2)$ is introduced to the objective

function, where λ>0 is the regularization parameter. The new regularized objective function is defined by:

$$Minimize\ J = \frac{1}{2}\sum_{(i,j)\in S} e_{ij}^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2 =$$

$$\frac{1}{2}\sum_{(i,j)\in S}\left(r_{ij-}\sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2 + \frac{\lambda}{2}\sum_{i=1}^{m}\sum_{s=1}^{k} u_{is}^2 + \frac{\lambda}{2}\sum_{j=1}^{n}\sum_{s=1}^{k} v_{js}^2$$

Each model has to be tested over different values of λ in order to find the best value that gives the most accurate predictions. For that reason, the method cross-validation is used. Cross-validation is a technique for assessing how the results of a statistical analysis will generalize to an independent data set. However, in big data problems cross-validation is very expensive to be used. As a result, the values of λ are usually not well-optimized [7].

There are many methodologies for the optimization problem described. Among them, the most popular are the stochastic gradient descent (SGD) and the alternating least squares (ALS) methods. Both of them they have their roots from the machine learning field. Although the SGD is an efficient method is sensitive to the initialization values and the value with which the step is defined. ALS is considered to be more stable [7]. Furthermore, ALS is inherently parallel and can deal with implicit ratings [26].

4.2.4 Alternating least squares

The optimization problem described is not convex. The minimization principle of alternating least squares is to keep one matrix fixed while calculating the other and vice versa [26]:

- In the first step, we keep the U fixed. We solve for each of the n rows of V a least-squares regression problem. For each row, only the observed ratings can be used. Let $\overline{v_j}$ be the jth row of V. The optimal vector $\overline{v_j}$ is determined by $\sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is}\cdot v_{js}\right)^2$ which can be seen as a least-squares regression problem. The features u_{j1}…u_{jk} are the constant values and the features v_{j1}… v_{jk} are the optimization variables. A total of n such least-squares problems are executed. These problems are independent between them so they can be parallelized.
- In the second step, we keep V fixed. . We solve for each of the m rows of U a least-squares regression problem. For each row, only the observed ratings can be used. Let $\overline{u_i}$ be the ith

row of U. The optimal vector $\overline{u_i}$ is determined by $\sum_{i:(i,j)\in S}\left(r_{ij} - \sum_{s=1}^{k} u_{is} \cdot v_{js}\right)^2$ which can be seen as a least-squares regression problem. The features $u_{j1}\ldots u_{jk}$ are the constant values and the features $v_{j1}\ldots v_{jk}$ are the optimization variables. A total of n such least-squares problems are executed. These problems are independent between them so they can be parallelized.

The pseudocode for the alternating least squares is:

```
1: procedure ALS(R, k, λ; X, Y )
2:     X ← 0, Y ← random initialization
3:     repeat
4:         for row u ← 1,m do
5:             xu ← (Y ⊤ Y + λI)−1Y ⊤ ru
6:         end for
7:         for column i ← 1, n do
8:             yi ← (X⊤X + λI)−1X⊤ ri
9:         end for
10:    until reached max iterations
11: end procedure
```

These two steps are repeated until convergence. This method has become famous in the last ten years because it combines good scalability and accuracy [20]. One benefit of this approach is the use of regularization. Without regularization, the algorithm may overfit. A common fix for this problem is to use the Tikhonov regularization, which penalizes large parameters [29]. The value of λ can be fixed across all the independent least-squares problem or it can be different for each row. The optimal value of λ is determined by trial and error. Typical values of k are between 20 and 200 [22].

Advantages and Disadvantages of ALS

The main advantages of alternating least squares are:

- It can be massively parallelized. Although the stochastic gradient descent is easier and faster than ALS, it cannot be parallelized efficiently. On the other hand, Most of ALS's computations are independent to each other.
- The second major advantage of ALS is that it can handle implicit data efficiently. The stochastic gradient descent approach cannot iterate through sparse tensors efficiently.
- The algorithm is flexible in facing multiple different data aspects and other application-specific requirements [20]

The main drawbacks of this approach are:

- The computations of ALS need parallel sparse manipulation. This kind of manipulation is challenging in order to achieve high performance because of the fact there is imbalanced workload, random memory access and task dependency [26].
- It may need many iterations to converge.
- It may give worse result than the stochastic gradient descent in large-scale problems with explicit ratings.

5 Goal of Dissertation

This dissertation is focused on the Collaborative Filtering methods. Three algorithms are implemented and evaluated. Both the categories of Collaborative Filtering are examined. The categories are the neighborhood-based and model-based methods. The Neighborhood-based category is examined both in the user-based and item-based approach. The similarity between the users or items is calculated with the Pearson coefficient or the adjusted cosine similarity. The similarity coefficients are used for both the two approaches. The third algorithm to be implemented is the Alternating Least Squares (ALS). It belongs to the latent factors family of model-based collaborative filtering.

The main goal of the dissertation is the parallelization of these algorithms. Parallelization is the most basic and efficient technique to reduce the computation time of algorithms that can be implemented on parallel. Not all the algorithms are able to be parallelized. Modern recommender systems demand fast results in order that the customers to be satisfied. There will be an examination on how the algorithms can be executed on parallel and how this approach can help recommender systems to be faster and give quicker results to customers.

The suggestions of Recommender Systems need to be accurate too. A system that can give fast results but with low accuracy is not sufficient. The very existence of recommender systems is their ability to provide accurate and robust results. A happy customer is more likely to be loyal to the system. As a result, the algorithms are going to be examined in terms of accuracy.

All the algorithms are implemented with the TensorFlow API using the python programming language. TensorFlow is a very popular framework and it is used widely in the scientific and commercial field. It provides helpful tools and has a big community of users that support it.

The algorithms will be evaluated in terms of accuracy and performance. According to the literature, the best ways to measure the accuracy are with the mean absolute error (MAE) and the root mean squared error (RMSE). There are many disagrees on which of them is the best. Both of them are used widely in the research and production field. The main difference between them is that MAE gives equal weight to all errors, while RMSE penalizes variance because it gives errors with large absolute values more weight than errors with small absolute values. By implementing both of them, there is a clearer picture on how models are affected by the existence of outliers or not.

The algorithms are going to be evaluated according to their performance too. The metrics to be used for that reason are the speedup and efficiency. Speedup is the measure which quantifies the performance gain from a parallel computation of a fixed-size application over its computation on a single core. Efficiency is the measure of the time percentage for which a machine is usefully employed in parallel computing. It is defined as the ratio of speedup to the number of processors.

It is clear, that when both of the two ways of evaluation are measured a safer conclusion can be conducted on which algorithm is the best. Accuracy is in a constant conflict with performance. All the algorithms are executed many times with different parameters in order to find the best combination that gives good accuracy and relatively best execution time.

The initial dataset is divided into two different datasets, the training and test dataset. For that reason, a number of random ratings are removed from the initial dataset. These ratings make up the test dataset. The ratings left are the training dataset. The ratio of the test ratings in the total number of ratings has to be small in order that the system can provide accurate and robust results. The ratio of the test ratings in the total ratings of all the experiments is decided to be 10%. There is no guarantee that all the users and items exist in both the train and test datasets. There is a purpose behind this implementation. For example, a situation where an item exists only in the test dataset can be interpreted as a new item entering the system. These kind of situations are very common in real recommendation systems and it is crucial to evaluate the system in these conditions.

5.1 TensorFlow

TensorFlow is an open source software library for high performance numerical computation used for both research and production. It is originally developed by engineers working for the Google Brain team. It is used for both research and production at Google [24]. TensorFlow has a flexible architecture that allows implementation across a variety of platforms. The supported platforms are CPUs, GPUs, and more recently TPUs. A TPU is a programmable AI accelerator designed by Google in order to provide high and efficient low-precision computation. TensorFlow is not only used by Google in the production area but there is a large number of companies that have projects based on this platform. Examples of them are Nvidia, Intel, AMD and eBay.

TensorFlow is a C++ based deep learning framework and is open sourced under the Apache 2.0 license. The stable APIs that are currently supported are the Python and C API and without API

backwards compatibility the C++, Go, Java, JavaScript and Swift API. The core of TensorFlow is implemented in C++ programming language, however the main language used is Python which currently supports the most features [23].

TensorFlow uses data flow graphs for numerical computations. The computational graph consists of nodes, which represent numerical operations, and edges which represent tensors. A Tensor is a multi-dimensional array used for computations [24]. Nodes take inputs, process them and give outputs. Tensors can be passed from one node to another. The edges control the flow of computation [23]. TensorFlow uses sessions for graph computations. Before the execution, the computational graph is empty and there are no variables in it. The session interprets user commands to initialize variables and build the computational graph [23]. The next step is the order of computation to be determined by creating the queue of nodes without dependencies. Then, the program executes the nodes in the queue in a way to ensure decreasing the unresolved dependencies until whole graph is computed [23]. One strong aspect of TensorFlow is distribution that can be implemented in a way that each node is assigned to a specific device for computation rather than running the whole graph in a single device. TensorFlow supports a GPU implementation. However, not all operations have GPU implementation. The supported GPUs are specific NVIDIA cards, using the appropriate version of the CUDA toolkit.

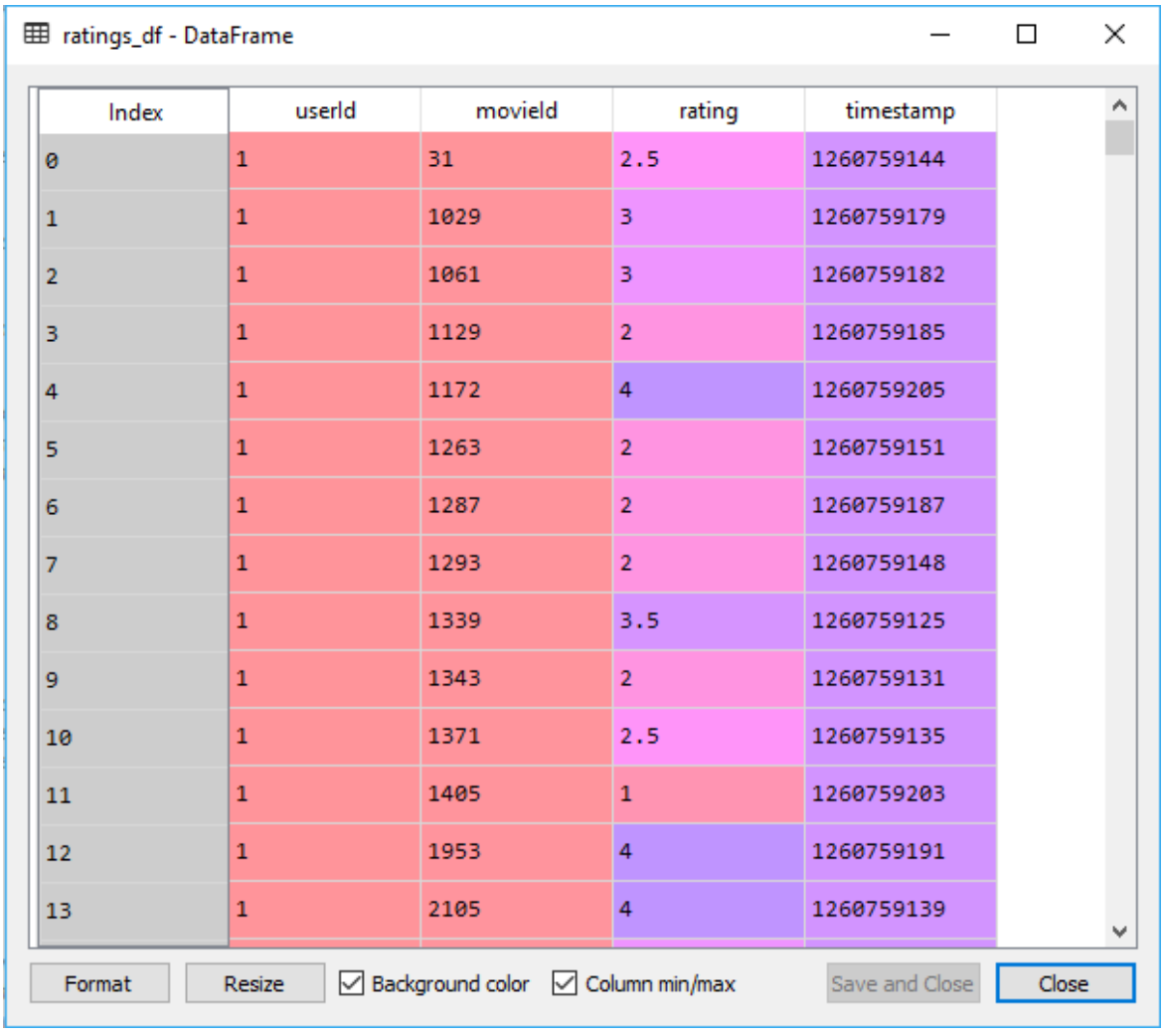
The main advantages using TensorFlow are [23]:

- *The computational graph model:* It is easy to visualize the problem and debug it.
- *Simple but strong Application Programming Interface (API):* It can be used both by experienced engineers in large companies and entry-level researchers.
- *Flexible architecture:* It can currently be implemented on CPUs, GPUs and TPUs.
- *High Distributed processing and performance:* Multiple engineers have evaluated it and concluded that TensorFlow has relatively better scalability compared with other tools. Also GPU platform has a much better efficiency than many-core CPU.

5.2 Dataset used and data preprocessing

The dataset used for this project is the small dataset provided by the GroupLens organization. The specific dataset is recommended for educational and research projects. It contains a subset of 671 users and 100004 ratings. The users were selected randomly. Each user in the subset has at least 20 ratings. No demographic data from the users are included. Each user is represented by an ID number

and nothing else is provided. The movies provided are 9066. Only movies with at least one rating are included in the dataset. However, the movie IDs are not sequential. For example, the movie ID with the maximum value is 193948 where the number of unique movies is only 9066. This can cause serious problems to the implementation of the algorithms in terms of computation speed and efficiency and memory overflow. Furthermore, the IDs from both the users and movies start from 1. This count is in conflict with the programming way of numbering an array. In most programming languages, the first position of an array is declared as 0. In order to solve this problems, both the movie and user IDs are normalized to begin from 0. In addition, the movie IDs are mapped in the range of the number of the unique movie IDs in the dataset which is 9066. The ratings provided are included in the ratings.csv file. A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file represents the following format:

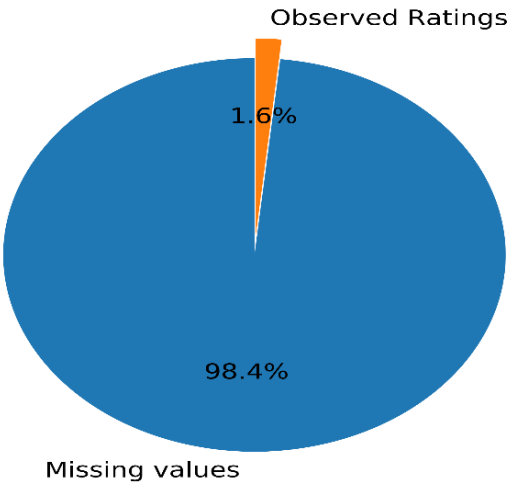


Index	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3	1260759179
2	1	1061	3	1260759182
3	1	1129	2	1260759185
4	1	1172	4	1260759205
5	1	1263	2	1260759151
6	1	1287	2	1260759187
7	1	1293	2	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2	1260759131
10	1	1371	2.5	1260759135
11	1	1405	1	1260759203
12	1	1953	4	1260759191
13	1	2105	4	1260759139

5.3.1 Structure of the ratings.csv file

Ratings are made on a 5-star scale, with half-star increments (0.5 stars - 5.0 stars). Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. For the purpose of this project, only the first three columns will be used.

The ratings matrix has 100004 observed ratings. The number of total ratings is the product of the number of users and items. That makes the missing values to be 5938282. The observed ratings makes only the 1,64% of the total ratings. That makes the ratings matrix to be extremely sparse.



5.3.2 The ratio between observed and missing values

The ratings matrix R can be represented by two implementations. The first implementation consists of a dense tensor where the missing values can be represented with 0 or -1. This implementation is very expensive for the system in terms of RAM storage and computational speed. The tensor needs to exist in the RAM during the whole execution of the algorithms. This can result in RAM overflow. In such cases, the hard drive is used reducing dramatically the efficiency of the algorithms. The second implementation uses the coordinate list method (COO). COO stores only the list of (row, column, value) tuples that represent the observed values of the rating matrix. Ideally, the entries are sorted first by row index and then by column index, to improve random access times. In this type of implementation, the ratings matrix is stored way more efficiently and the RAM allocation is significantly lower than the first approach. TensorFlow has an implementation of the COO method. It represents a sparse matrix as a tf.SparseTensor. This sparse tensor is consisted of three separate dense tensors: indices, values, and dense_shape. TensorFlow has special operations for this kind of tensor in order to decrease the duration time of executions and RAM size. Examples of such operations are multiplication and addition between sparse tensors.

System

The System that was used to perform the experiments is a desktop personal computer. Specifically, the central processing unit (CPU) is the Ryzen 2700x by AMD. This CPU has 8 cores and 16 logical processors. The basic clock speed from the CPU cores is 3700 MHz. The system has a single Graphics Processing Unit (GPU) the GeForce GTX 1060 with 3 GBs of GDR5 VRAM. This GPU consists of 1152 CUDA cores. The clock speed is 1708 MHz. The GPU is listed on the official CUDA-enabled GeForce products and has a compute capability of 6.1. The random access memory (RAM) of the system is the G.Skill Ripjaws of 16 GB capacity. The type of the RAM is DDR4 with base speed 3200 MHz.

6 Implementation of algorithms

6.1 Neighborhood-based Models

6.1.1 Pearson Correlation

The Pearson correlation is the most common measure for neighborhood-based collaborative filtering and it is used by many Recommender Systems. As the literature suggests, neighborhood-based methods is better to be divided in the offline and online phase. With this approach, computations are isolated and it is easier to evaluate them separately.

Offline Phase

The correlations between all the users of the system are computed in the offline phase. This is the heaviest task in terms of computing time. It is very important to find out which are the independent computations of the algorithm. Then, these independent computations can be parallelized to reduce the overall computation time and increase the performance of the model. Each calculation of finding the similarity between two different users is independent from the others. This leads to a good level of parallelization since the main computations in the offline phase are the similarities calculations.

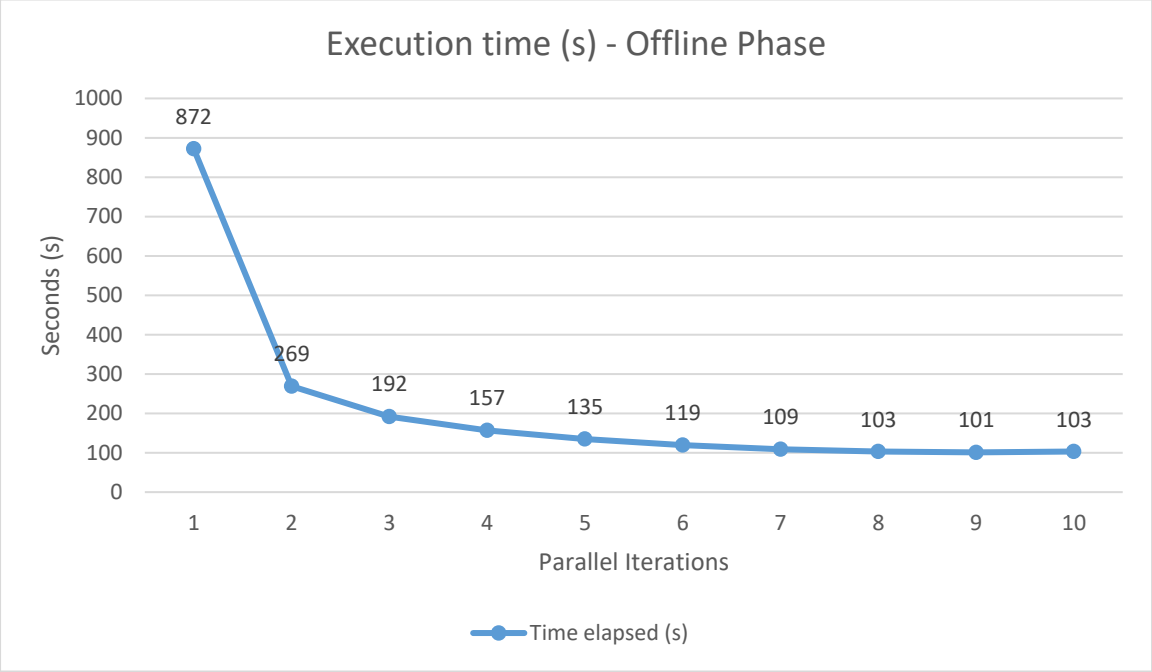
To begin with, a tensor of size (number of users, number of users) is created. This tensor contains consecutive numbers in ascending order starting from one. It represents the total number of parallel computations. Then with the help of the operation tf.map_fn we parallelize the calculations. The simplest version of tf.map_fn repeatedly applies the callable function to a sequence of elements from first to last. This operation has a parameter called parallel_iterations that enables the tuning of the amount of threads to be used in this operation. If the computations inside the tf.map_fn operation are independent, the same results should be given for parallel_iterations = 1 and parallel iterations > 1. It is crucial to state that only half of the total similarities need to be computed because the ratings matrix R is symmetrical. The Pearson similarity between two same entities equals to 1.

The computation of mean values of μ_u and μ_v between two users, can be implemented in different ways which lead to different results. With the first way which is simpler and computationally less costly, the mean is calculated for all the user's observed ratings on items. In the second approach, the mean is computed only with the ratings of the set which contains the mutual rated items. There

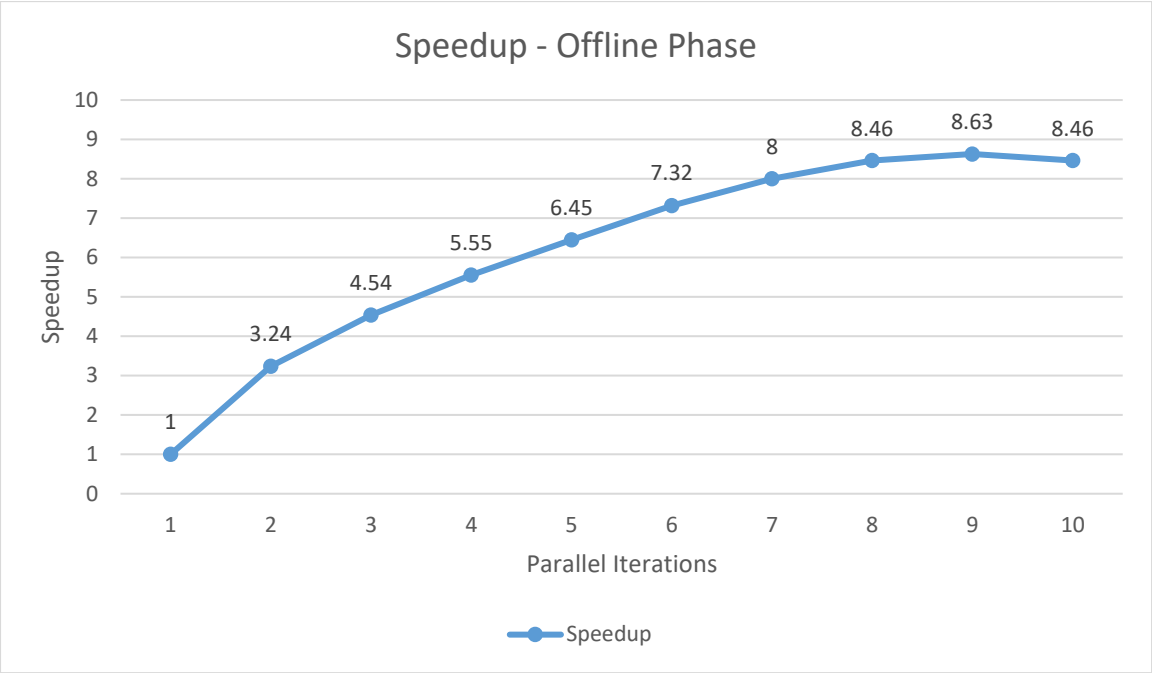
is not a clear answer on which method provides better results. The original neighborhood-based algorithm with the Pearson similarity implements the second method. This is also the method used in this dissertation.

The results of the offline phase which is the correlation matrix among the users of the system is saved outside the TensorFlow session as a CSV file. This file is going to be used in the online phase of the algorithm in order to predict the missing values of the ratings matrix.

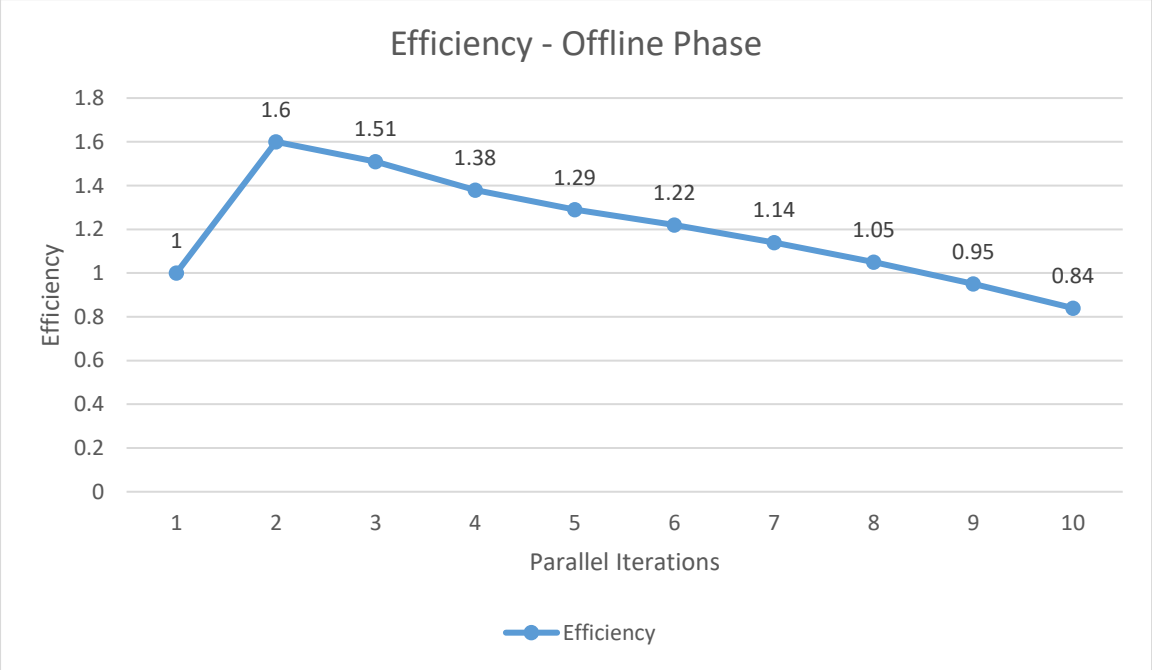
The offline phase of the algorithm can be evaluated only in terms of performance. The below charts summarize the performance of the offline phase of the algorithm according to the number of parallel iterations specified.



This diagram shows the duration of the different execution times of the offline phase according to the number of parallel iterations. The best time achieved when the parameter was set to 9.



The speedup metric shows that the system responds positively in the increase of the parallel iterations parameter until it converges when it takes the value 9. The system is scaling up until this point.

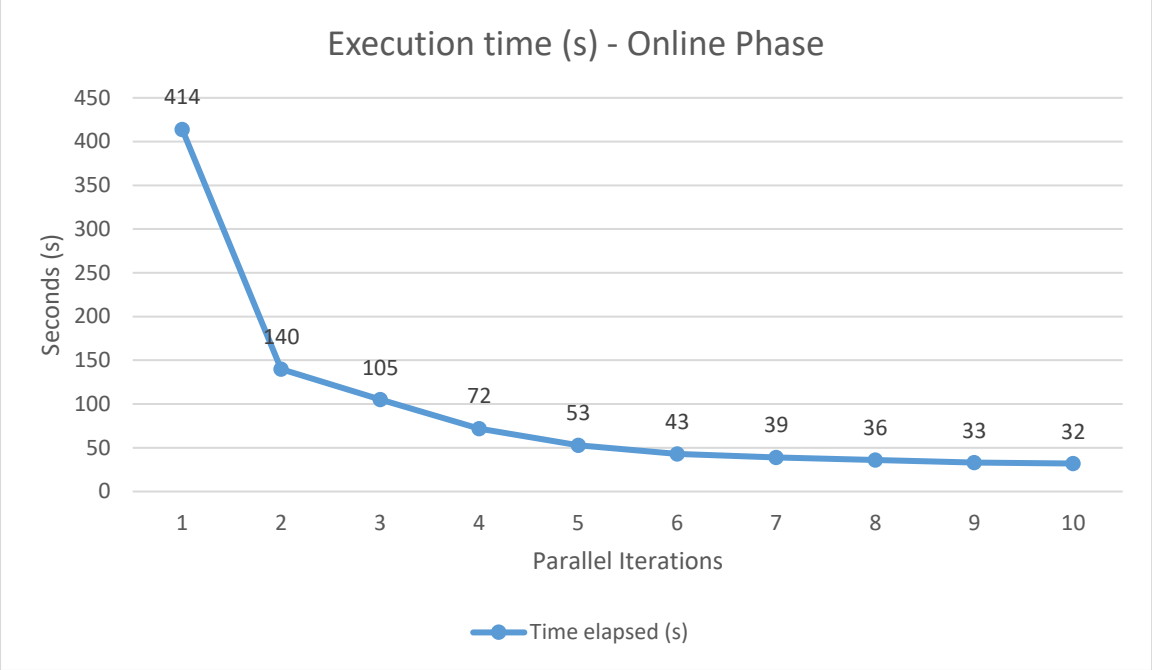


This diagram depicts how efficient the system is when more resources are available. The conclusion of the three diagrams is that the system is the fastest when the parallel iterations are set to 9. The biggest impact on the system was the increase of parallel iterations from 1 to 2.

Online Phase

This is the phase where the missing values are predicted. The correlations among all the users that calculated in the offline phase are used for that reason. The computation of each predicted rating $\widehat{r_{ij}}$ is independent from the others. As a result, all these computations can be parallelized. The parameter closest users is used in order to define the size of each neighborhood. An increase to this parameter results to better accuracy and higher computational time. In real recommender systems, the online phase is calculated independently for each user for efficient reasons. Each time a user requests a recommendation, the system provides back his recommendations. In this dissertation, for evaluation reasons the whole ratings matrix is calculated. The online phase of the algorithm is evaluated in terms of accuracy and performance.

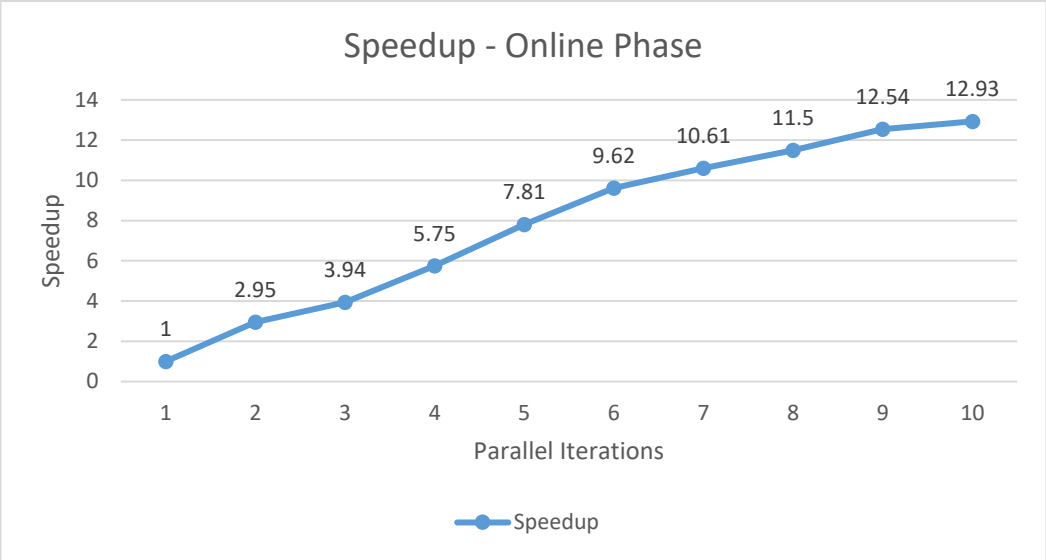
In order to evaluate the performance of the online phase, we define the size of neighborhood to be 10. This value is stable for all the experiments.



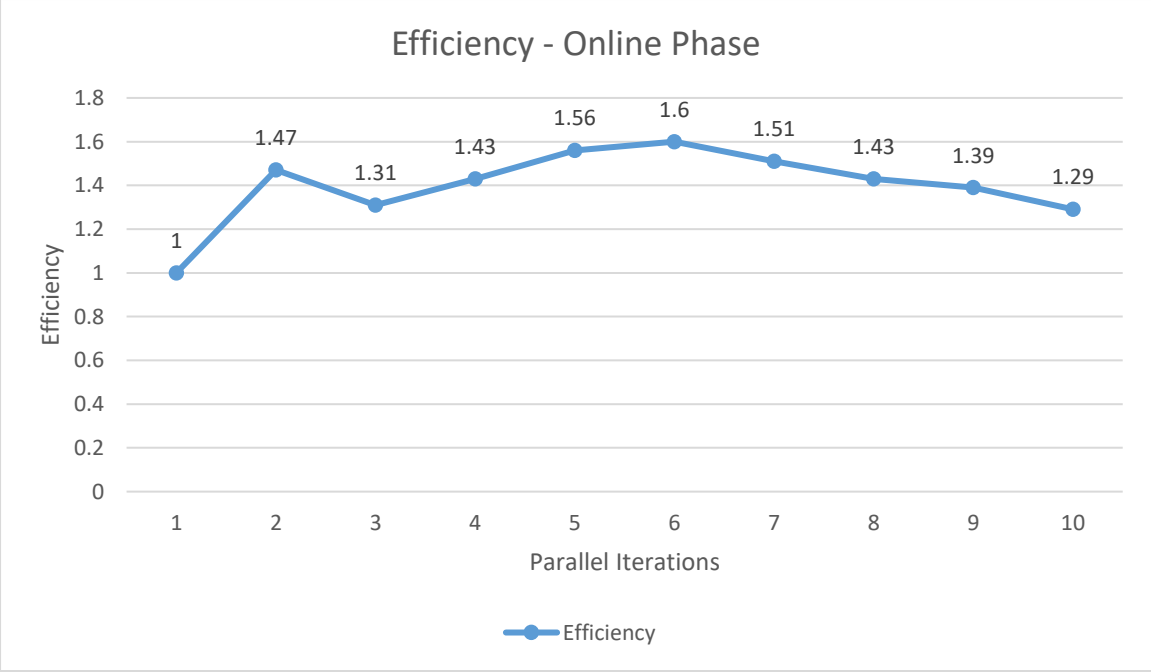
It is clear that the execution time needed is much less than the offline phase. This is expected because the computation time for the offline phase is $O(m^2 \cdot n')$ while for the online phase is $O(k \cdot m)$. The fastest execution time is observed when the number of parallel iterations is 10. In this situation, there is a full utilization of the system.

One important notice with the implementation of the neighborhood-based algorithms is that there is not a good exploitation of the GPU. Many of the operations used have not a GPU implementation.

This causes a bottleneck to the system because there has to be a constant communication between the CPU and GPU. As a matter of fact, when the program runs on both the CPU and GPU the performance results are worse. For this reason, CPU is the only platform chosen the program runs on.



The speedup results show that the online phase is highly scalable. As the system gives more resources, the computation time keeps reducing at a satisfactory level. The results concerning the level of parallelization are better than the offline phase.

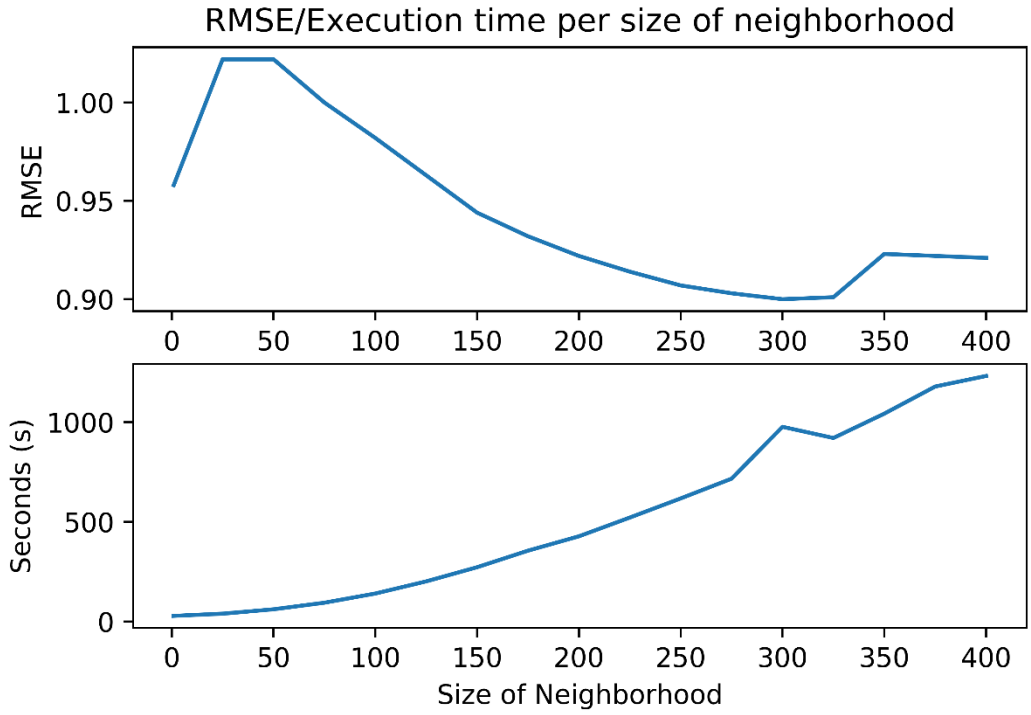


The mean absolute error (MAE) and root mean squared error (RMSE) metrics are used to evaluate the algorithm in terms of accuracy. The below diagrams show how they are affected when the size of user neighborhood is increased. The diagrams depict also the correlation between the size of neighborhoods and the execution time. For this experiments, the system is fully utilized and all its sources are available.

As expected, MAE most of the times is being reduced when the size of user's neighborhood increases. An interesting notice is that until the size of the neighborhood be 50 the results are getting worse. The best results in terms of accuracy happen when the number of closest users inside a neighborhood are close to 300.

As the neighborhood increases, the execution time increases dramatically. The computation time needed when the best accuracy is achieved is 1000 seconds. A fully utilized system needs 101 seconds to calculate the correlations between all the users in the system. That is 10% of the time needed in order that the same system predict all the values of the ratings matrix with best accuracy. Because of real recommender systems do not predict all the ratings at once (they recommend items only to the users requested them), this situation may not be a problem. However, the calculation of specific recommendations needs to be examined by real recommender systems. According to the results of that examination, decisions have to be taken about the accuracy/performance sacrifice.

Recommender systems need to be as accurate and fast as possible in order that the customers be satisfied.



The results of the RMSE metric are smoother than the MAE's results. As the diagram depicts, the best results are achieved when the size of user' neighborhood is close to 330. As we can see, an increase to this size does not always mean better accuracy. An increase to the size of user's neighborhood after the 330 value make worse the results of RMSE.

6.1.2 Adjusted Cosine Similarity

Offline Phase

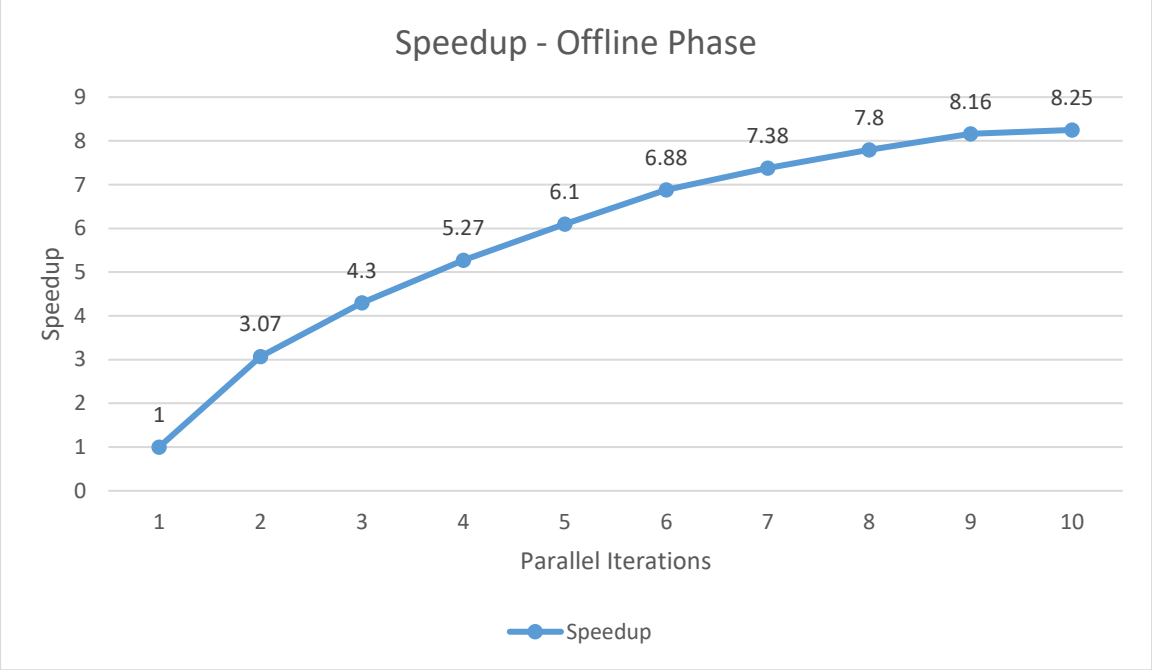
The correlations between all the items of the system are computed in the offline phase. This is the heaviest task in terms of computing time. Each calculation of finding the similarity between two different items is independent from the others. This leads to a good level of parallelization since the main computations in the offline phase are the similarities calculations.

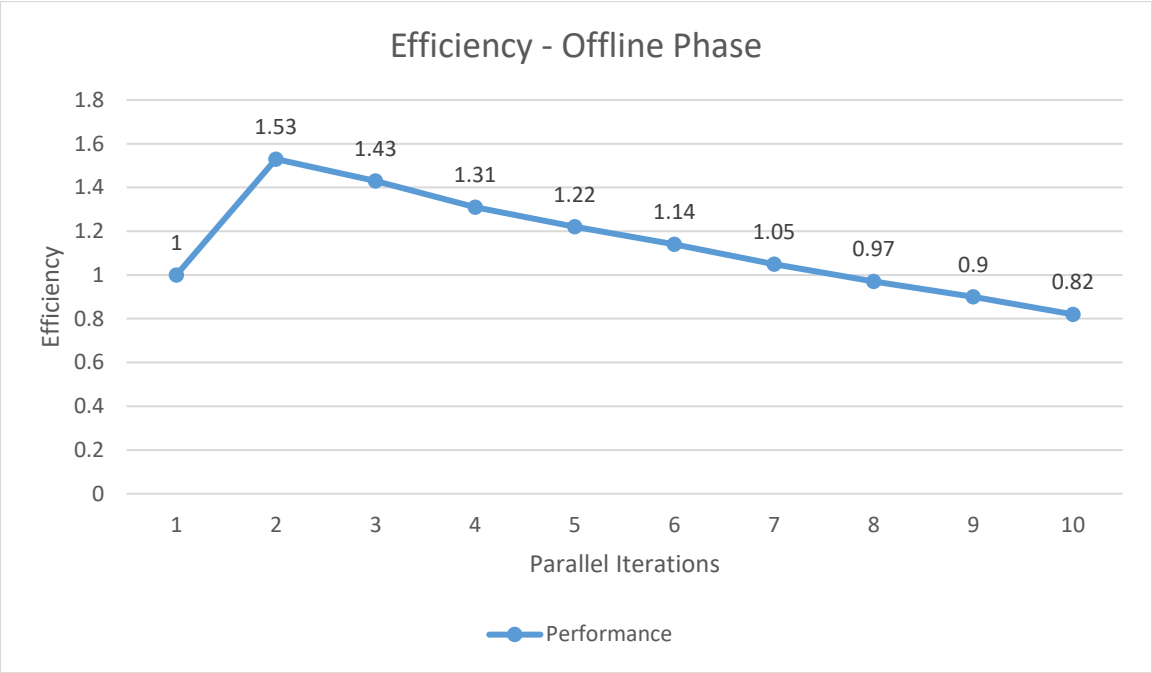
Initially, a tensor of size (number of items, number of items) is created. This tensor contains consecutive numbers in ascending order starting from one. It represents the total number of parallel computations. Then with the use of the function tf.map_fn all the correlations between the items are calculated on parallel.

The means of users are constantly used during the execution of the program. For this reason, they are pre-computed before the algorithm starts in order to be achieved better performance. The computation of users' means are also implemented on parallel.

The results of the offline phase which is the correlation matrix among the items of the system is saved outside the TensorFlow session as a CSV file. This file is going to be used in the online phase of the algorithm in order to predict the missing values of the ratings matrix.

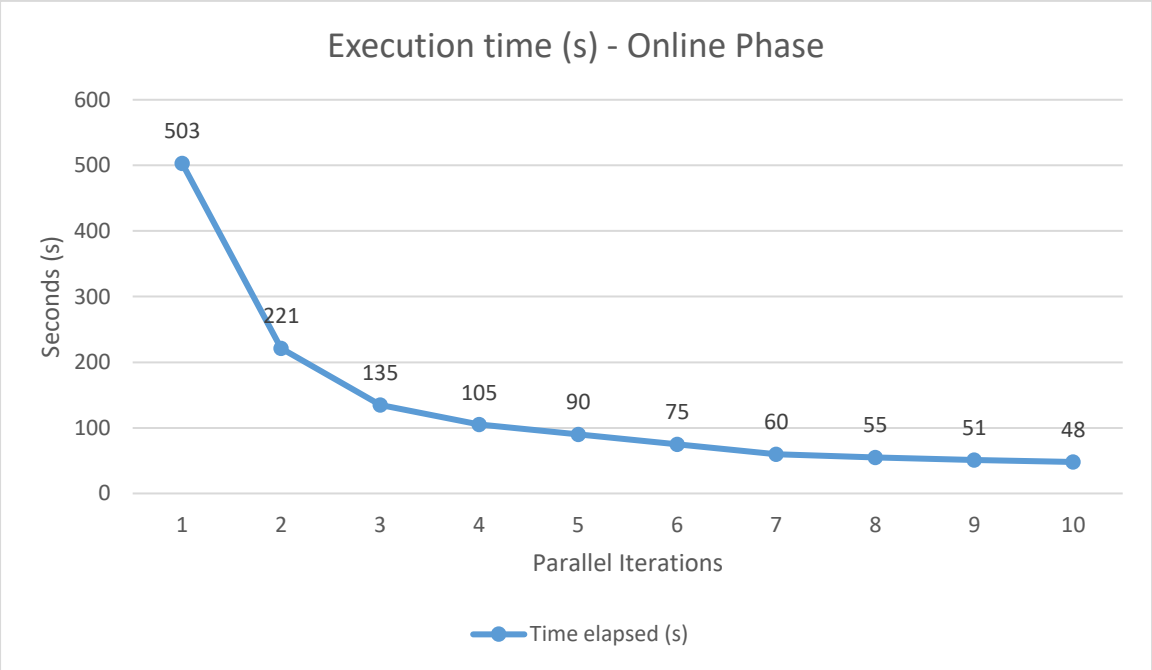
The offline phase of the algorithm can be evaluated only in terms of performance. The computation time needed is much bigger than the computation of the user-based neighborhood algorithm. For practical reasons the evaluation of the algorithm is conducted on the first 5 rows of the ratings matrix.

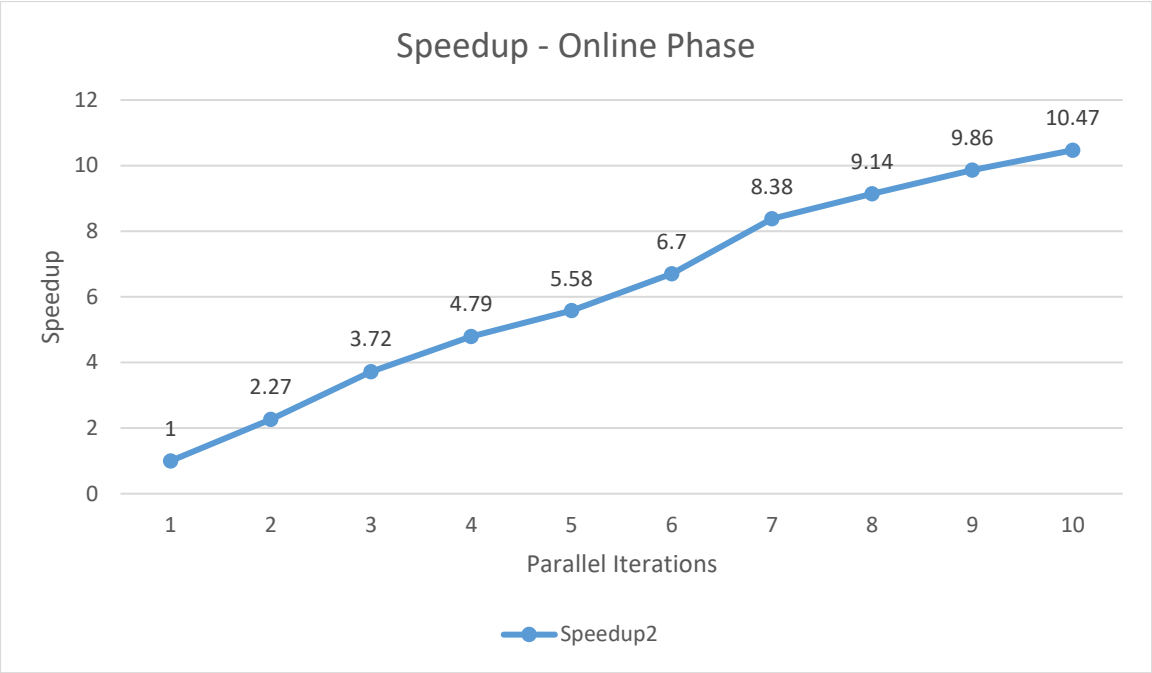




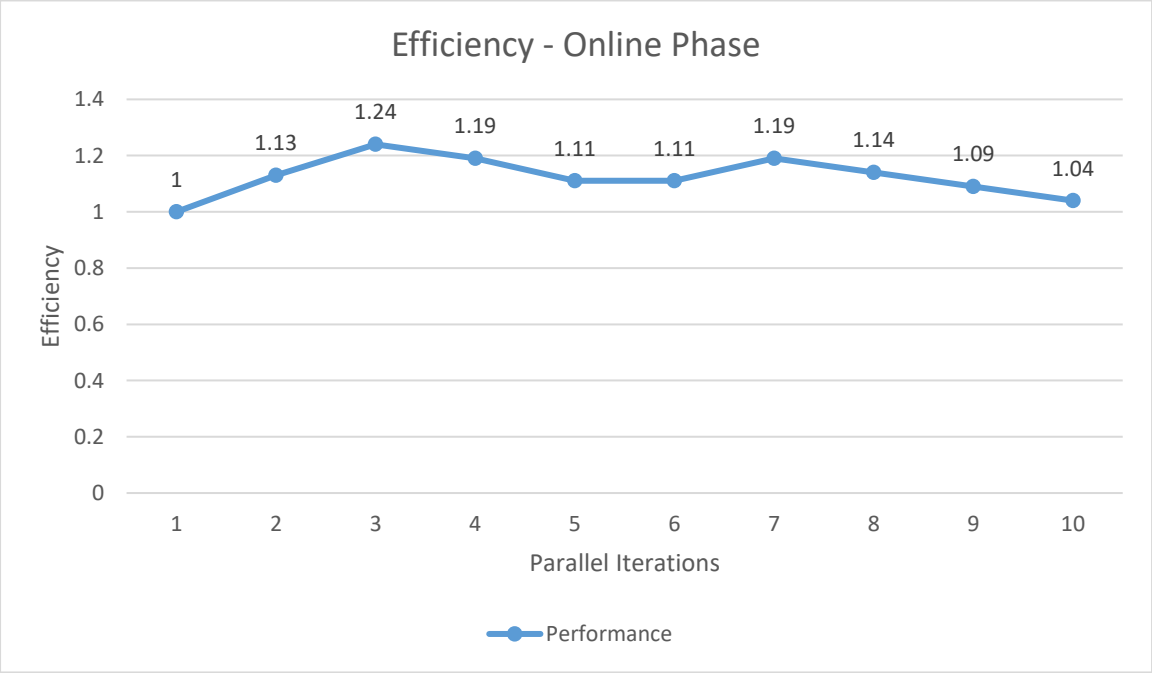
Online Phase

The computation time of the online phase is tiny compared to the offline phase. There is a good utilization of the system. The computation time is similar to the online phase of the user-based algorithm.

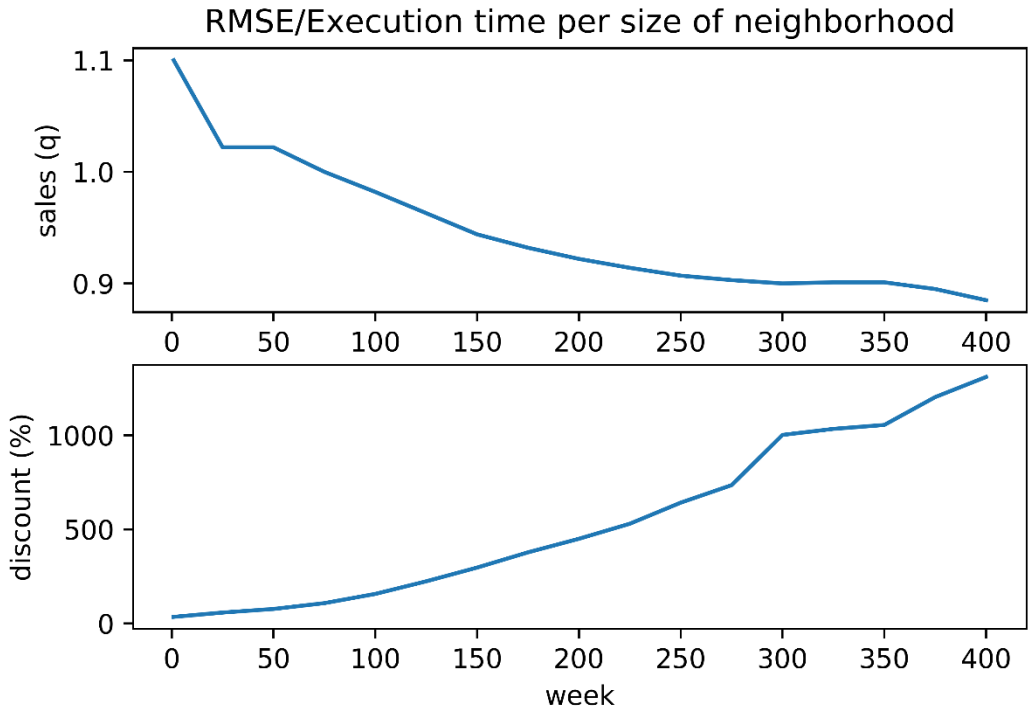




We can see there is a good parallelization of the algorithm. In every step there is a scale in its performance.



The efficiency measures are worse compared to the online phase of the user-based algorithm.



The item-based neighborhood algorithm achieved the best results in RMSE. The best accuracy achieved was 0.88

6.2 Model-based Models

6.2.1 Alternating least squares

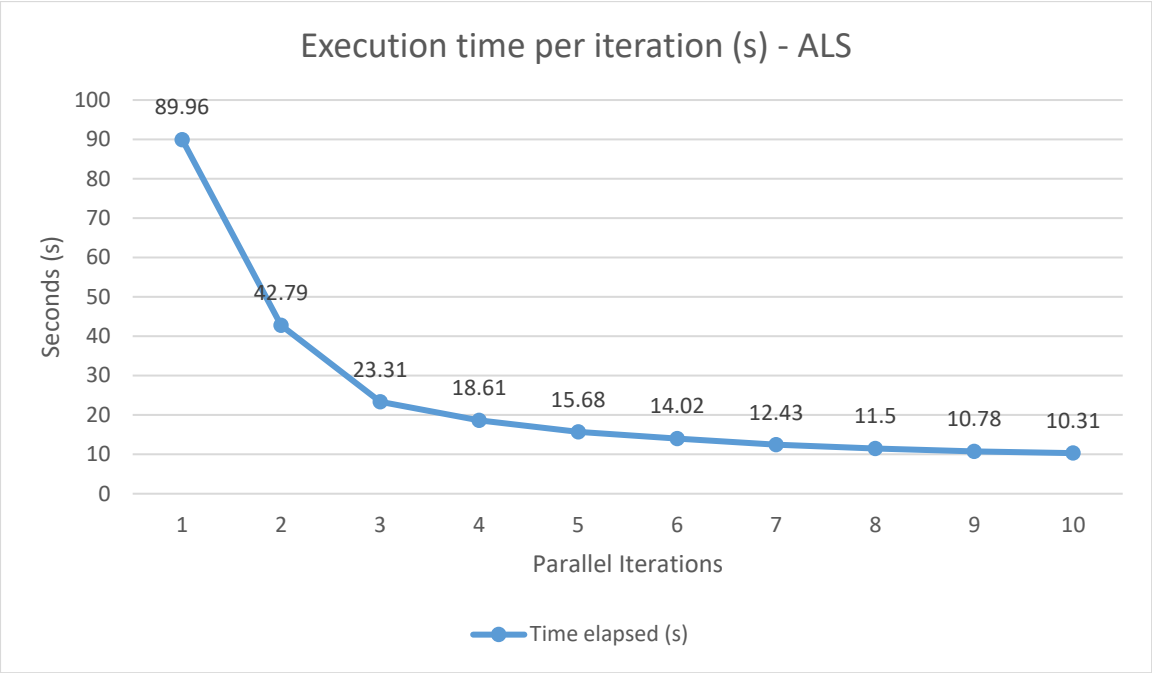
Alternating least squares (ALS) is one of the most popular collaborative filtering algorithms. It is considered to be the state of the art of today's recommendation algorithms. It was researched a lot in the past years and it has been improved significantly. Many of the modern recommender systems are based on this approach. The strong feature of this algorithm is its efficiency. ALS is highly parallelizable which results to relatively low computational times. It belongs to the latent factors family of algorithms. ALS is parallelized by parallelizing the updates of U and V which are independent from each other. The optimal value of λ is determined by trial and error. Once the two matrices are calculated the approximate matrix R with the predicted ratings can be found in one shot. As a result there is no reason to divide the problem into two sub-problems like in the

neighborhood-based methods. The algorithm is examined in terms of accuracy and performance. The algorithm can be tuned by selecting the number of factors to be used and the parameter λ. The initial parameter is multiplied in each row with the number of movies each user has rated.

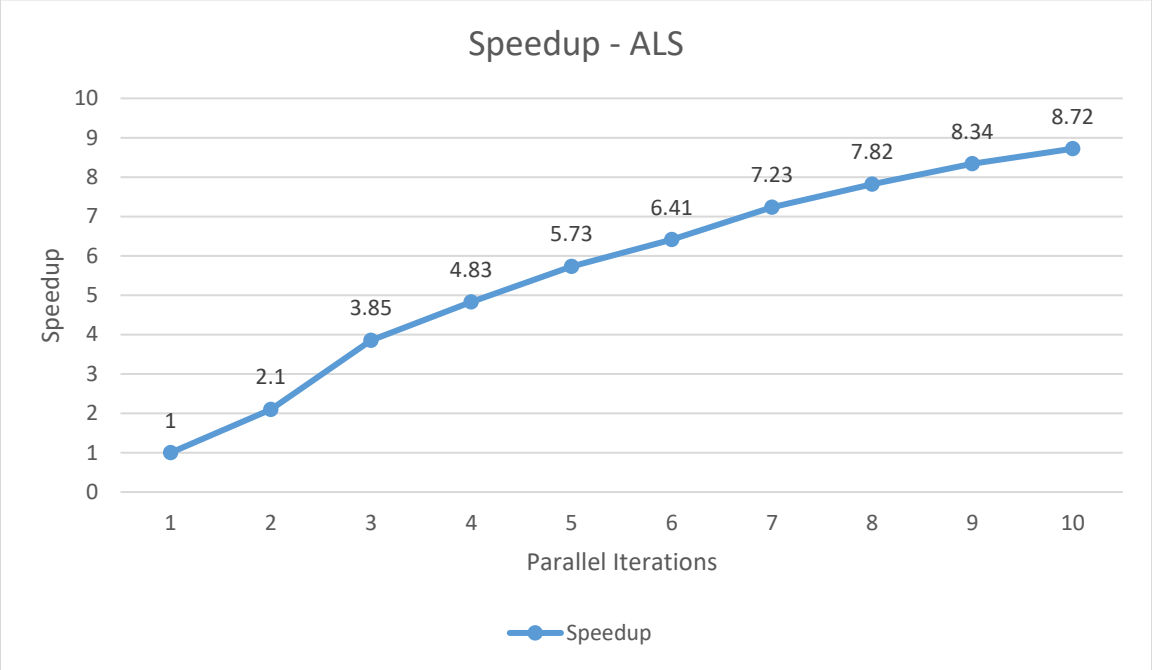
Initially the two matrices are randomly initialized with values having 0 mean and standard deviation of 0.2. In order to update the matrices four methods are called in each iteration. All of these methods are fully parallelized with the help of the tf.while_loop TensorFlow function. This function allows the parallelization of independent computations inside a tf.Session. By setting the parameter parallel_iterations, there is control on how many parallel iterations are computed simultaneously. The first method updates the rows of the U matrix and the second one updates the rows of the V matrix. After these two steps, some rows of the matrices are filled with zeros. This situation occurs when users or items exist on the test set but they are not in the train set. In order to solve this problem, two other methods are implemented. Each one of them calculates the mean of the columns of U and V respectively and applies these means to the rows that their values equal to zero. This procedure is computed on parallel.

The ALS algorithm is an optimization algorithm that improves its results in each iteration until it converges. The accuracy of its results are examined initially per iteration. Then, it is examined how the accuracy changes when the algorithm is being tuned by the number of factors used.

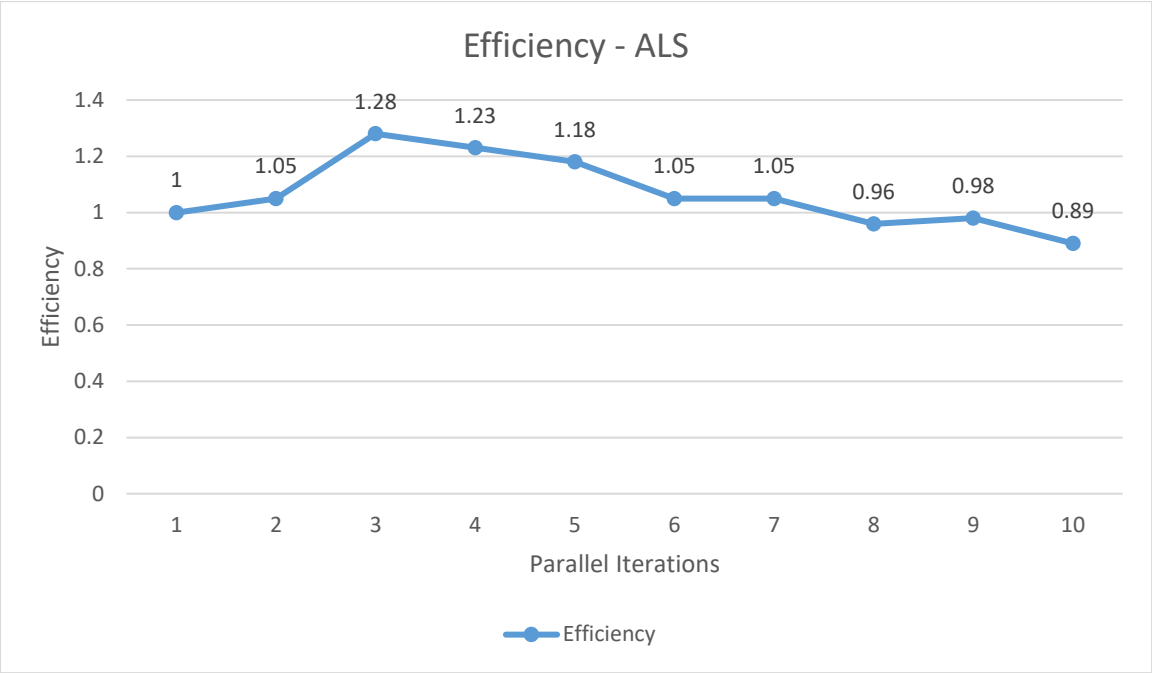
Finally, there is an evaluation on the performance of the algorithm in terms of speedup and efficiency. The evaluation is made on the computation of one iteration.



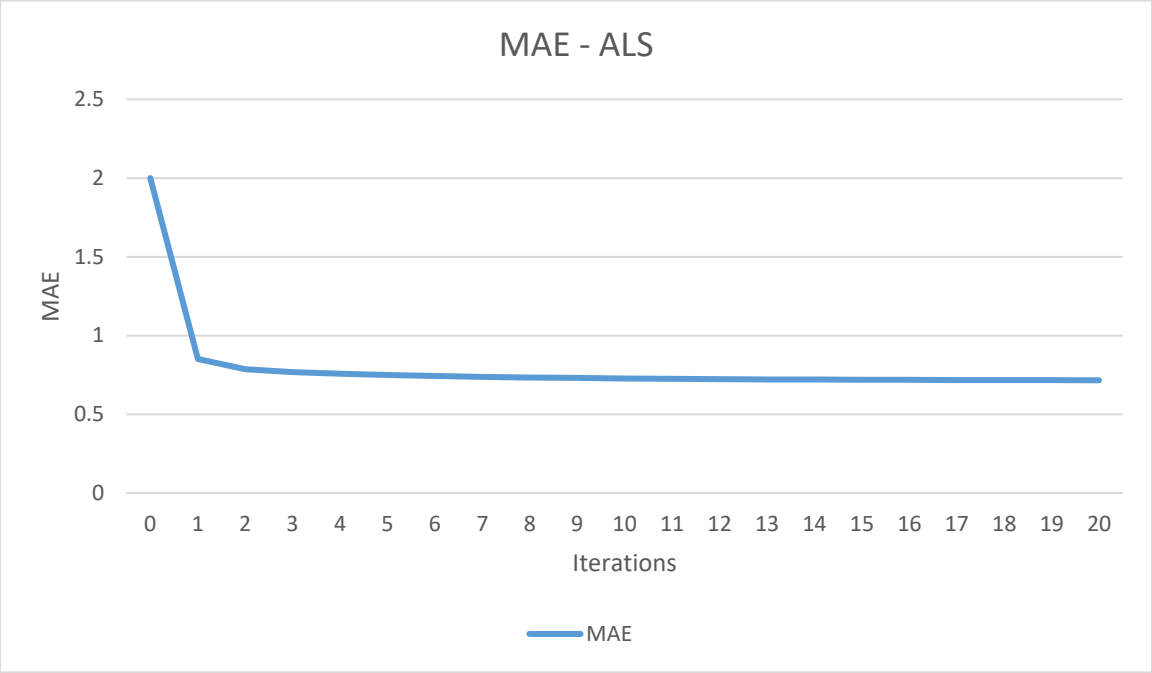
As the system gives more resources, the computation time keeps reducing at a satisfactory level.



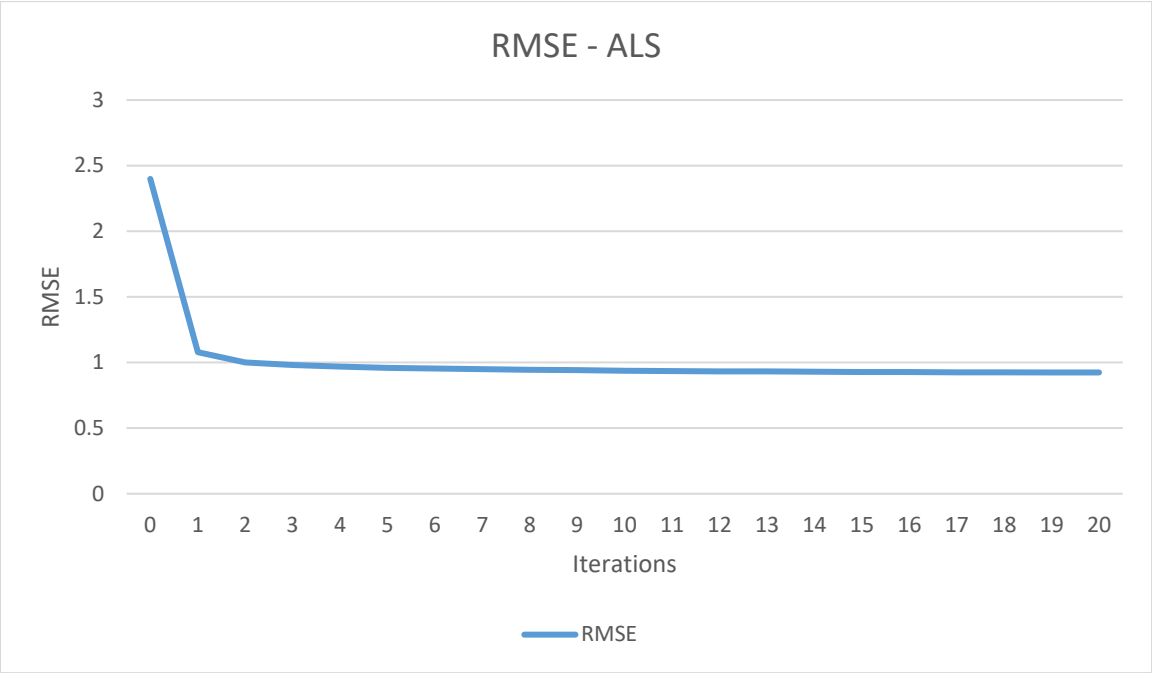
As the diagram depicts, ALS is able to scale up very well. The algorithm keeps scaling up until the number of parallel iterations se to 10.



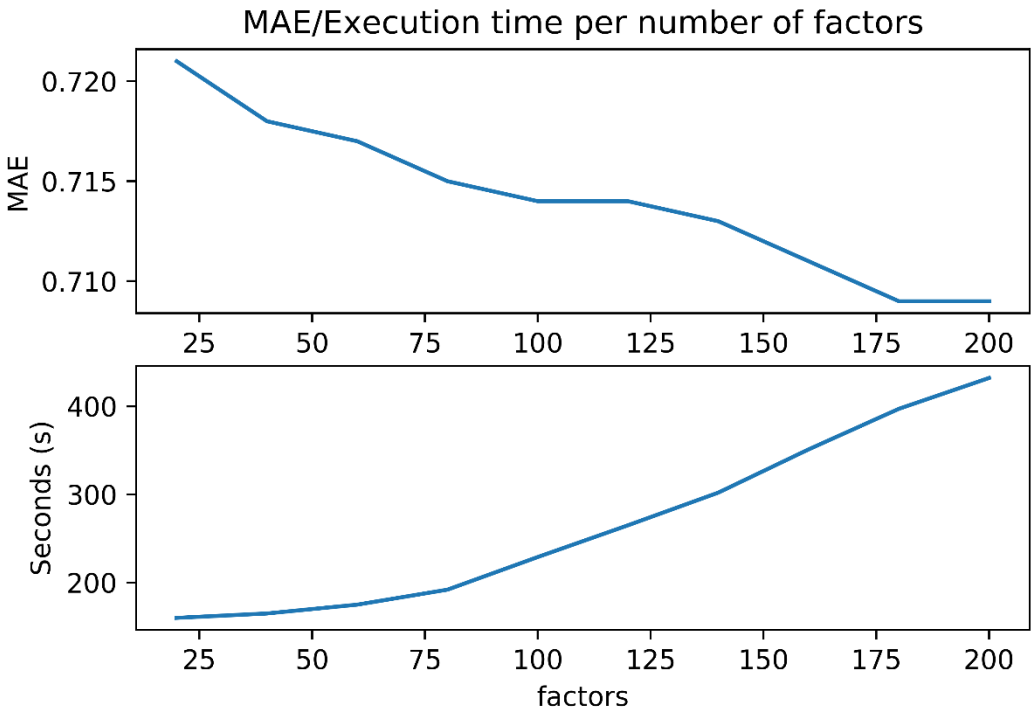
The efficiency results of ALS in terms of performance are worse than those of neighborhood-based methods. However, there is a satisfactory level of efficiency for each new parallel iteration.



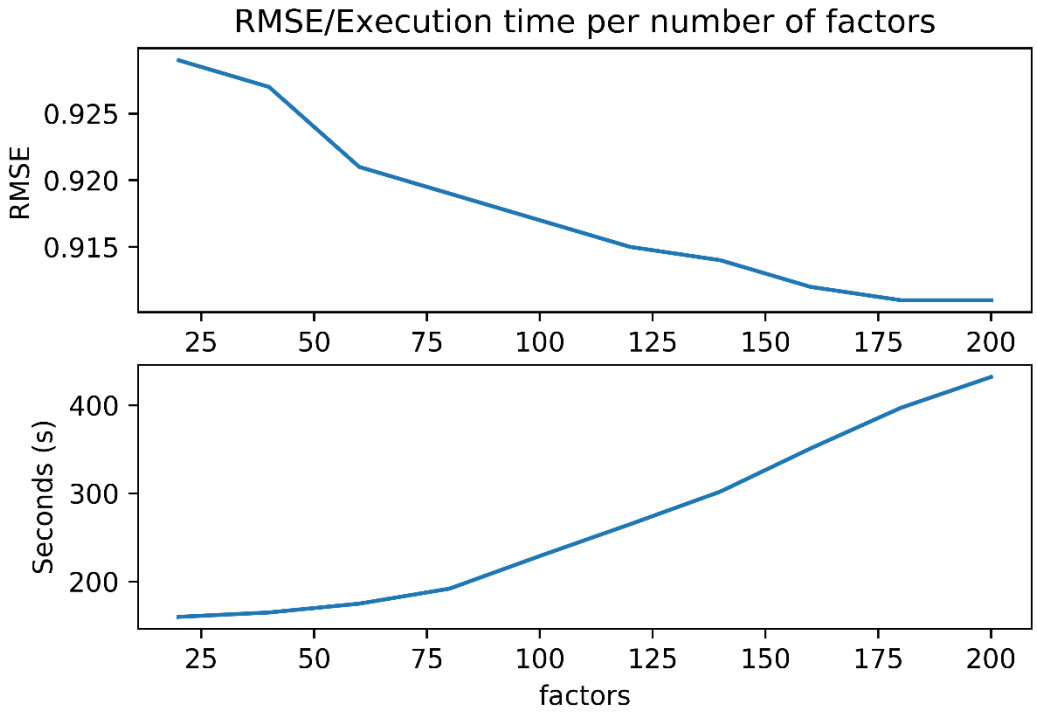
The diagram shows that the algorithm converges in the 16th iteration. The biggest improvement was seen from the first to second iteration. A satisfactory accuracy can be achieved in the earliest iterations of the algorithm.



The rate of RMSE change is slower than MAE's. We can see that the RMSE is getting stable after the 11th iteration of the algorithm.



As we can see, the MAE metric does not improve enough as more factors are set. On the other hand, the execution time is increased dramatically making the use of more factors inefficient.



The same results are given by the RMSE metric. The overall improvement of RMSE from 20 to 200 factors used is just 0.01. This improvement comes with a great expense in terms of computation time.

7 Conclusions and Future Work

All the algorithms have the possibility to be parallelized. The experiments that conducted concerning their performance show that they were implemented on parallel efficiently. The most accurate algorithm in predicting the missing values was the item-based neighborhood with the adjusted cosine similarity while the fastest was the alternating least squares.

Neighborhood-based algorithms

The neighborhood-based methods provided the most accurate results. The best results in terms of accuracy are achieved by the item-based neighborhood algorithm with the adjusted cosine similarity. However, the offline phase computation time for this algorithm was eight hours when the utilization of the system was at 100%. The accuracy that provides is not efficient regarding to the execution time needed. The dataset consists of 671 users and 9066 items. The ratio between users and items is 7.4%. This is not the case for real recommender systems where the size of users is usually bigger than the size of items. The offline computation times for the user-based and item-based algorithms used are $O(m^2 \cdot n')$ and $O(n^2 \cdot m')$ respectively. That is the reason, the offline phase of the item-based algorithm is slow.

The user-based neighborhood algorithm with the Pearson similarity has the second highest accuracy between all the algorithms implemented. It is very efficient compared to the item-based approach. Its offline phase can be computed relatively fast in a way that it can be implemented in real recommender systems.

In the designing face of neighborhood-based systems, there must be a careful thought on which approach will be implemented. According to the dataset's characteristics (size of users and items), the appropriate method should be implemented. Both of them, when they used correctly can provide relatively accurate results.

Alternating Least Squares algorithm

The ALS algorithm has the lowest results in terms of accuracy. However, it is the fastest algorithm implemented. It can converge fast and its accuracy results are not far away from those of the neighborhood-based methods. With a small sacrifice in accuracy is the most efficient algorithm implemented. That is the reason, most of the real recommender systems are based on it.

Future Work

The algorithms that they were implemented are on their basic form. In order to achieve higher results in terms of accuracy, more sophisticated forms are needed to be implemented. Future work concerns deeper analysis of particular mechanisms such as probabilistic methods.

When the complexity of the algorithms is increased by increasing the values of algorithm's parameters (e.g. number of factors, size of neighborhood), there is a need of more RAM. Usually, the RAM which is available on the GPUs is limited compared to the RAM of the system. It is observed, that the automatic mechanism TensorFlow has for distributing the parallel iterations between the CPU and GPU is not optimal. For that reason, there is a need of manual placement of operations across the system to achieve better performance results.

TensorFlow is a powerful tool with many capabilities. When the problem size is getting big more resources are needed in order to execute the algorithms. TensorFlow supports clustering where a set of CPUs and GPUs are used in order to solve the problem. Clustering is considered to be the best way to face big data problems. The next step of this dissertation is the distributed implementation of the algorithms.

8 References

[1] Kunal Shah, Akshaykumar Salunke, Saurabh Dongare and Kisandas Antala. *Recommender systems: An overview of different approaches to recommendations*. International Conference on Innovations in Information, 2017

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 2005

[3] Sanjeevan Sivapalan, Alireza Sadeghian, Hossein Rahanam and Asad M. Madni. Recommender Systems in E-Commerce. World Automation Congress, 2014

[4] Qian Zhao, F. Maxwell Harper, Gediminas Adomavicius and Joseph A. Konstan. *Explicit or implicit feedback? engagement or satisfaction?: a field experiment on machine-learning-based recommender systems.* SAC, 2018

[5] Aggarwal and Charu C. *Recommender Systems The Textbook*, 2016

[6] Sarika Jain, Anjali Grover, Praveen Singh Thakur, Sourabh and Kumar Choudhary. *Trends, problems and solutions of recommender system*. International Conference on Computing, Communication & Automation, 2015

[7] Ricci, F., Rokach, L., Shapira, B. and Kantor. *Recommender Systems Handbook*, 2011

[8] Badrul M. Sarwar, George Karypis, Joseph Konstan and John Riedl. *Recommender Systems for Large-scale E-Commerce: Scalable Neighborhood Formation Using Clustering*. 5th International Conference on Computer and Information Technology (ICCIT), 2002

[9] Manolis G. Vozalis and Konstantinos G. Margaritis. *Analysis of Recommender Systems' Algorithms.* The 6th Hellenic European Conference on Computer Mathematics & its Applications, 2003

[10] T. Chai1,2 and R. R. Draxler. *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature, 2014*

[11] C. J. Willmott and K Matsuura. *Advantages of the Mean Absolute Error (MAE) over the Root Mean Square Error (RMSE) in Assessing Average Model Performance*, Climate Research, 2005

[12] Leily Sheugh and Sasan H. Alizadeh. *A note on pearson correlation coefficient as a metric of similarity in recommender system*, AI & Robotics 2015

[13] Xiaodong Zhang and Yong Yan. *Modeling and characterizing parallel computing performance on heterogeneous networks of workstations*. Seventh IEEE Symposium on Parallel and Distributed Processing, 1995

[14] Xingfu Wu and Wei Li. *Scalability of parallel algorithm implementation.* Proceedings Second International Symposium on Parallel Architectures, 1996

[15] S. Prakash and R. Bagrodia. *An adaptive synchronization method for unpredictable communication patterns in dataparallel programs*, Proceedings of 9th International Parallel Processing Symposium, 1995

[16] Chandima Hewa Nadungodage, Yuni Xia, John Jaehwan Lee, Myungcheol Lee and Choon Seo Park. *GPU accelerated item-based collaborative filtering for big-data applications*, 2013 IEEE International Conference on Big Data, 2013

[17] Wei Tan, Liangliang Cao and Liana Fong. *Faster and Cheaper: Parallelizing Large-Scale Matrix Factorization on GPUs,* Cornell University, 2016

[18] Hao Li, Kenli Li, Jiyao An and Keqin Li. MSGD: *A Novel Matrix Factorization Approach for Large-Scale Collaborative Filtering Recommender Systems on GPUs*, IEEE Transactions on Parallel and Distributed Systems, 2018

[19] Badrul Sarwar, George Karypis, Joseph Konstan and John Riedl. *Item-based collaborative filtering recommendation algorithms*, Proceedings of the 10th international conference on World Wide Web, 2001

[20] Yehuda Koren, Robert Bell, Chris Volinsky. *Matrix Factorization Techniques for Recommender Systems*, Computer, 2009

[21] Manda Winlaw, Michael B Hynes, Anthony Caterini, Hans De Sterck. *Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark*, IEEE 21st International Conference on Parallel and Distributed Systems, 2015

[22] Yifan Hu, and Chris Volinsky. *Collaborative Filtering for Implicit Feedback Datasets*, 2008 Eighth IEEE International Conference on Data Mining, 2008

[23] Damir Demirović, Emir Skejić and Amira Šerifović–Trbalić, *Performance of Some Image Processing Algorithms in Tensorflow*, 25th International Conference on Systems, Signals and Image Processing (IWSSIP), 2018

[24] Hao Chen. *Spam Message Filtering Recognition System Based on TensorFlow*, 3rd International Conference on Mechanical, 2018

[25] F. Maxwell Harper and Joseph A. Konstan. *The MovieLens Datasets: History and Context*, ACM Transactions on Interactive Intelligent Systems, 2015

[26] Jing Chen, Jianbin Fang, Weifeng Liu, Tao Tang, Xuhao Chen and Canqun Yang. *Efficient and Portable ALS Matrix Factorization for Recommender Systems*, IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017

[27] Gábor Takács and Domonkos Tikk. *Alternating least squares for personalized ranking*, 2017

[28] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan. *Large-Scale Parallel Collaborative Filtering for the Netflix Prize*, Lecture Notes in Computer Science book series, 2008