— A Parallel Recommender System Using a Collaborative Filtering Algorithm for Movie Recommender System —

Projna Saha
School of Computer Science
Carleton University
Ottawa, Canada K1S 5B6
projnasaha@cmail.carleton.ca

December 18, 2021

Abstract

Recommendation systems are most commonly used to recommend items for web users. Collaborative filtering or recommender systems use a database about user preferences to predict additional topics or products a new user might like. Recently, collaborative filtering algorithm is widely used in recommendation systems. However, in some practices, the computational time of CF may be unsatisfactory. Meanwhile, in some cases there are noises in data,i.e., some data are invalid, it also has a great impact on algorithm performance. In this research, we employed SPARK Framework for making parallel movie recommender system. The experiment on two datasets consisting data collected from actual social networks (i.e., MovieLens-100k & Netflix-6.2M) were conducted. To build this platform, we used Databricks Community Edition that allows to make clusters. From the analysis of the results we found the overall parallel performance and computational time for the recommender system from Ganglia UI. The experimental results showed that for social networks application, the proposed system could effectively improve the computational time and achieve satisfactory performance through invalid data existed.

1 Introduction

In the past decades, recommender systems have gotten a lot of attention. A recommender system is sometimes known as a recommendation tool that takes the information about the user as input and finds a similarity based on the user's "rating" or "preference" for an item. For example, most famous websites such as YouTube, Netflix, IMDb, MovieLens etc. are using some sort of recommendation systems that provide recommendations of the movies, shows that are similar to the ones that have been watched in the past [1]. There are majorly five types of recommender systems that work primarily in the Media and Entertainment industry: Collaborative Recommender system, Content-based recommender system, Community-based recommender system, Knowledge-based recommender system, and Hybrid recommender system [2]. Hence, Collaboration filtering, content-based filtering, and hybrid filtering are the three most used conventional techniques.

The idea behind a Content-based (cognitive filtering) recommendation system is to recommend an item based on a comparison between the content of the items and a user profile.

In simple words, viewers may get recommendations for a movie based on the description of other movies [3]. On the other hand, collaborative filtering mimics user-to-user recommendations. It predicts user preferences as a linear, weighted combination of other user preferences. Collaborative filtering recommendation algorithm has many advantages, such as high speed, high efficiency, and good robustness, and the recommendation results are more accurate than other algorithms. Content-based filtering can recommend a new item but needs more data of user preference in order to incorporate the best match. Collaborative filters are expected to increase diversity because they help us discover new products. Though collaborative filtering (CF) is one of the most popular algorithms due to its simplicity, it consists of noises in data, i.e., data scarcity, poor accuracy, scalability, and cold start problems and this suffers overall algorithm performance [4].

To reduce the data noise and speed up the time, Sun & Luo [5] developed a novel parallel recommending system based on CF with correntropy. Instead of traditional measures used in recommendation algorithms, the correntropy was employed to compute the similarity of two items or users to achieve insensitive performance to outliers. Furthermore, to reduce the computational cost, Spark framework is a good choice for parallel processing. This paper aiming to evaluate parallel performance using three different algorithms in the SPARK framework and compare the accuracy model (i.e., MSE, MAE, RMSE) measured from the recommender system. The result shows a comparative analysis between two datasets to measure the parallel performance.

2 Literature Review

2.1 Similarity Computation

In general, CF algorithms can be classified into two categories: neighborhood-based approaches and model-based methods. Though the neighborhood-based collaborative algorithm is gaining popularity for its simplicity, stability, and efficiency methods, it spends lots of time obtaining the recommendation list even though it is faster compared with other algorithms. The most popular technique to create a recommender system is to use CF through neighborhood-based interpolation, and a vital stage is called "neighborhood selection," which is closely tied to the similarity weights measure. There are some methods for calculating similarity based on ratings, and if there is any faulty or incorrect data, their performance will quickly deteriorate [6].

2.1.1 Cosine Vector (CV) Similarity

Despite that user-based collaborative filtering is the most successful technology for building recommender systems to date and is extensively used in many commercial recommender systems, unfortunately, the computing complexity of these solutions grows linearly with the number of customers, which can be several million in typical commercial applications. [7]. Cosine similarity is a useful metric for determining how similar two documents are in terms of a particular topic. In the realm of data mining, the approach is also used to measure cluster cohesion. From the equation, u and s two instances and cosine similarity calculates the distance between their shape vector X_u and X_s. Using r_{ui} to define the rating from user u to item i. This can be obtained $X_{ui} = r_{ui}$ if user u has rated item i, and $X_{ui} = 0$ in other cases. The similarity between two users u and s could be computed by:

$$\cos(u,s) = \cos_{us} = \frac{\sum_{i\epsilon I_{us}} r_{ui} r_{si}}{\sqrt{\sum_{i\epsilon I_u} r_{ui}^2}\sqrt{\sum_{i\epsilon I_s} r_{si}^2}} \tag{1}$$

Where r_{ui} denotes the rating value from user u to item i, and the similar definition applies to r_{si}. In addition, I_{us} denotes the items that both user u and user s rated, the same conception holds true for I_u and I_s.

2.1.2 Pearson Correlation (PC) Similarity

The Pearson correlation coefficient indicates how strong a linear link exists between two variables. Even though CV similarity does not account for differences in the mean and variance of the evaluations, a more popular method, known as PC similarity, was devised. It can solve the problems that CV similarity causes and produce better outcomes [8]. The following is the similarity measure [9]:

$$PC(u,s) = PC_{us} = \frac{\sum_{i\epsilon I_{us}} (r_{ui} - \overline{r_u})(r_{si} - \overline{r_s})}{\sqrt{\sum_{i\epsilon I_{us}} (r_{ui} - \overline{r_u})^2}\sqrt{\sum_{i\epsilon I_{us}} (r_{si} - \overline{r_s})^2}} \tag{2}$$

Where $\overline{r_u}$ denotes the average rating value of user u, and $\overline{r_s}$ denotes the average rating value of user s. It should be noted that the standard deviation of the ratings in evaluations is based on the common items rather than the entire set of items, and z-score normalization may be necessary. The sign of this similarity denotes whether the correlation between two users or two items is direct or inverse, and the magnitude denotes the connection's strength.

2.1.3 Spearman Correlation (SC)

When evaluating correlations involving ordinal variables, Spearman correlation is frequently utilized. It is another way of calculating similarity by using the rank of those ratings in place of actual rating value [10]. The equation becomes,

$$SC(i,j) = SC_{ij} = \frac{\sum_{u\epsilon U} (k_{u,i} - \overline{k_i})(k_{u,j} - \overline{k_j})}{\sqrt[2]{\sum_{u\epsilon U} (k_{u,i} - \overline{k_i})^2}\sqrt[2]{\sum_{u\epsilon U} (k_{u,j} - \overline{k_j})^2}} \tag{3}$$

Where $k_{u,i}$ denotes the rank of the rating of item i of user u and $k_{u,j}$ shows the rank of the rating of item j of user u. $\overline{k_i}$ and $\overline{k_j}$ calculate the average rank of item i and item j.

2.1.4 Adjusted Cosine (AC) Similarity

The adjusted cosine similarity measure is a modified version of vector-based similarity that accounts for the fact that various users have varying rating methods; in other words, some people may score objects highly in general while others may prefer to rank items lower. AC similarity is defined as [5],

$$AC(i,t) = AC_{it} = \frac{\sum_{u\epsilon U_{it}} (r_{ui} - \overline{r_u})(r_{ut} - \overline{r_u})}{\sqrt{\sum_{u\epsilon U_{it}} (r_{ui} - \overline{r_u})^2}\sqrt{\sum_{u\epsilon U_{it}} (r_{ut} - \overline{r_u})^2}} \tag{4}$$

Where U_{it} denotes the users who rated both items i and t. In addition, r_{ui} to define the rating from user u to item i.

2.1.5 JacRA Similarity

In collaborative filtering based recommendation system, cold start, scalability and data sparseness are important factors that affect its performance. Since some existing similarity computing methods are inaccurate in measuring similarity, and accuracy must be improved, a new similarity calculation method is proposed, which is called JacRA similarity [11]. Defining similarity between two users in JacRA as,

$$sim(u,v) = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} \cdot \frac{\sum_{i\epsilon I_u \cap I_v} \frac{min(r_{ui},r_{vi})}{max(r_{ui},r_{vi})}}{|I_u \cap I_v|} \tag{5}$$

Here, $r_{u,i}$ and $r_{v,i}$ are the ratings of users u and v to item i respectively, $I_u \cap I_v$ is the set of item which have been used by users u and v concurrently. $min(r_{ui}, r_{vi})$ is the minimum of $r_{u,i}$ and $r_{v,i}$. $max(r_{ui}, r_{vi})$ is the maximum of $r_{u,i}$ and $r_{v,i}$. Following the same way, similarity between items can be written as in JacRA as,

$$sim(u,v) = \frac{|U_i \cap U_j|}{|U_i \cup U_j|} \cdot \frac{\sum_{u\epsilon U_i \cap U_j} \frac{min(r_{ui},r_{uj})}{max(r_{ui},r_{uj})}}{|U_i \cap U_j|} \tag{6}$$

Here, U_i and U_j are the sets of users who have used item i and j, respectively. $U_i \cap U_j$ the set of users who have used item i and j concurrently. $U_i \cup U_j$ are the set of users who have used item i or j

2.2 Mean Squared Distance (MSD)

The mean squared distance (MSD) is a time-dependent measure of a particle's location divergence from a reference point. MSD evaluates the similarity as follows:

$$MSD(u,s) = MSD_{us} = \frac{I_{us}}{\sqrt{\sum_{i\epsilon I_{us}} (r_{ui} - r_{si})^2}} \tag{7}$$

In comparison to the Pearson Correlation (PC) method, the MSD method's application is limited because it does not capture negative correlations between two people or things [12]. Actually, the prediction accuracy might be improved when considering such negative correlations.

2.3 Rating Prediction

Rating prediction is a well-known recommendation task that attempts to forecast a user's rating for goods that the user has not yet evaluated. Users' explicit input, i.e. their past ratings on some goods, is used to calculate predictions [13]. The rating prediction system first finds the similarity among users using the previous data available in the data set. Using that similarity, the system predicts the ratings for different items for different users [14].

2.3.1 Weighted Average (WA)

The weighted average of neighborhood ratings is 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 a row-wise fashion, before determining the average rating of the neighborhood. The weighted average of these ratings is the predicted value and it is given by [15]:

$$\widehat{r}_{ui} = \frac{\sum_{j \epsilon N_u(i)} sim(i,j) r_{ju}}{\sum_j \epsilon N_u(i) |sim(i,j)|} \tag{8}$$

Where r_{iu} and r_{ju} are the ratings of items i and j given by user u and where $\widehat{r}_{ui}$ depicts the predicted value of item i of user u.

2.3.2 Mean-Centering (MC)

In moderated multiple regression models or polynomial regression models, mean centering has been proposed as a solution to problems of collinearity. Due to significant correlations between variables, mean centering does not minimize critical collinearity [16]. By comparing a rating to the mean rating, mean-centering is utilized to establish if it is positive or negative [17]. The equation becomes as the following in the prediction of rating using similarity value with mean centering approach. User-based prediction of a rating r_{ui} can be defined as:

$$\widehat{r}_{ui} = \overline{r_u} + \frac{\sum_{j \epsilon N_u(u)} sim(i,j)(r_{ju} - \overline{r_j})}{\sum_j \epsilon N_u(u) |sim(i,j)|} \tag{9}$$

Where $\overline{r_u}$ represents the average of average of the ratings given by user u to the items in i. This method can also be used for the item-based CF algorithm, and then the item-based predicted rating of r_{ui}.

$$\widehat{r}_{ui} = \overline{r_i} + \frac{\sum_{j \epsilon N_u(i)} sim(i,j)(r_{ju} - \overline{r_j})}{\sum_j \epsilon N_u(i) |sim(i,j)|} \tag{10}$$

Where $\overline{r_i}$ represents the average of the item ratings, i given by the user u.

2.3.3 Z-Score (ZS)

To handle the spread in the individual rating scales, z-score normalization is introduced in addition to the mean-centering method. In addition to the mean-centering method, z-score normalization is used to handle the spread in the individual rating scales [18]. In user-based CF algorithm, it can be obtained by normalizing the rating $\widehat{r}_{ui}$ by dividing the user-mean-centered rating by the standard deviation σ_u of the ratings given by user u:

$$\widehat{r}_{ui} = \overline{r_u} + \sigma_u \frac{\sum_{j \epsilon N_u(u)} sim(i,j)(r_{ju} - \overline{r_j})/\sigma_u}{\sum_j \epsilon N_u(u) |sim(i,j)|} \tag{11}$$

Similarly, in item-based CF algorithm, the Z-score normalization of $\widehat{r}_{ui}$ by dividing the item-meancentered rating by the standard deviation σ_i of ratings given to item i [19],

$$\widehat{r}_{ui} = \overline{r_i} + \sigma_i \frac{\sum_{j \epsilon N_u(i)} sim(i,j)(r_{ju} - \overline{r_j})/\sigma_i}{\sum_j \epsilon N_u(i) |sim(i,j)|} \tag{12}$$

2.4 Correntropy Similarity

In non-Gaussian noise situations, correntropy is a useful tool for studying higher order statistical moments. With real, complex, and quaternion data, correntropy has been applied [20]. Correntropy is a measure of how similar two random variables are in a neighbourhood dominated by a variable called kernel bandwidth. The kernel bandwidth also affects the "observation window" in which similarity is assessed, which could be an effective strategy to reduce the negative effects of inaccurate data [21–23]. A common form of correntropy between two variables P and Q is:

$$V_\sigma(P, Q) = E[K_\sigma(P - Q)] \tag{13}$$

Where $k_\sigma(.)$ is a kernel function that satisfies Mercer theory, and $E[K_\sigma(P - Q)]$ denotes the expectation operator of the kernel. It takes advantage of a kernel trick that nonlinearly maps the input space to a higher dimensional feature space. Generally, correntropy is robust against outliers. Due to the specific properties of correntropy, this method is been improved in the CF algorithm.

2.5 Spark Framework

Apache Spark is an open-source, Java virtual machine (JVM)-based cluster framework that used for big data workloads like Hadoop or Mesos. It utilizes in-memory caching and optimized query execution for fast queries against data of any size. It can also run in the cloud. However parallel computation offers a viable way to alleviate this problem [24]. Spark provides programmers with a resilient distributed dataset (RDD). The RDD is a read-only multiset of data items that is distributed throughout the cluster. As shown in Fig-1, in the first phase, RDD objects are constructed to generate directed acyclic graph (DAG) and operate it. In the second phase, a high-level DAG scheduler is responsible for splitting the DAG into several smaller stages of tasks and setting each task as ready condition. In the next phase, tasks are scheduled and distributed by cluster managers to parallel workers. And in the last stage, workers execute their tasks concurrently in individual threads or cores. Moreover, the RDD is the basis of Spark framework, and the whole operations in RDD support the entire Spark programming. The high-level Spark regards the language scale as the upper implementation of RDD, and it is believed the first system to use an interactive, general, efficient programming language to process a considerable scale of datasets on distributed machines [25]. Considering the advantages of Spark, in this article, Sun & Wang [5] developed a parallel personalized recommender system as shown in Figure-2, through the use of a Spark framework. Here, the basic module mainly handles XML file, calling the running methods in task modules and passing the parameters in the XML file to task modules in the form of map. There are other extra modules used to support the task modules and the basic modules such as evaluation, kernel, mean, similarity, CFLog and XmlParser. In the new proposed Spark Framework, CFOffLine & CFOnLine are the key parts where CFOffLine module calls Similarity module to calculate the corresponding similarity of the data with outliers, and CFOnLine module predicts the item scores based on some algorithms, e.g., k-nearest neighbors (k-NN) algorithm. The advantages of the aforementioned approach could achieve results with less computational efforts, which outperforms other traditional CF methods in computational time both on stand-alone and distributed computing environments. Specifically, when the size of the dataset is too big, this algorithm could achieve a more obvious advantage with the help

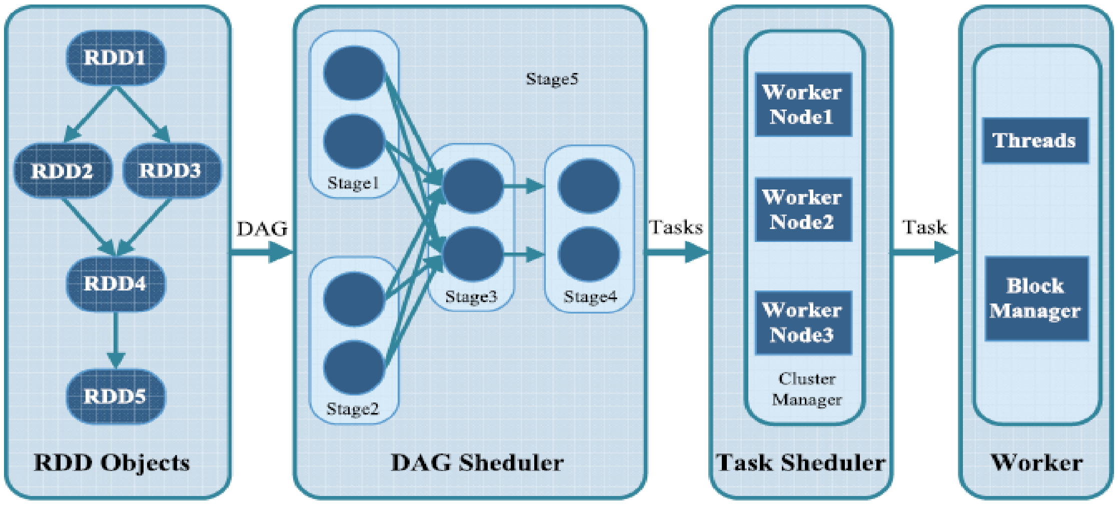


Figure 1: The task scheduling procedure in Spark. [5]

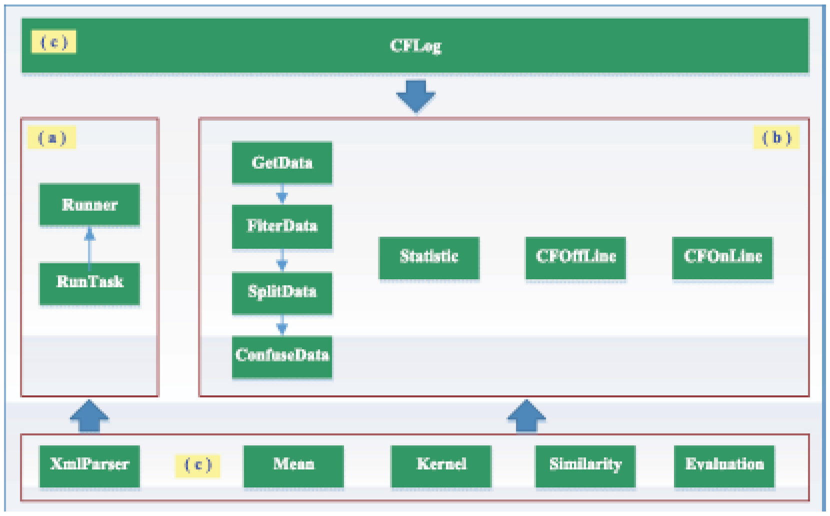


Figure 2: System modules in Spark framework. (a) Basic modules. (b) Task modules. (c) Extra modules. [5]

of parallelism computing mechanism. Furthermore, compared with other methods, this newly proposed correntropy-based approach may achieve better performance in addressing the datasets with outlier, since correntropy is insensitive to invalid data. When the size of invalid data is relatively big, newly SPARK framework may still perform better.

3 Research Methodology

3.1 Experimental Setup

3.1.1 Datasets

In this article, two real datasets are used in the experiments. For our own system, we use the open-source MovieLens dataset from GroupLens. This dataset contains 100K data points of various movies and users, containing 100000 ratings of 1682 movies made by 943 users. Another dataset Netflix-6.2M from Kaggle, contains 6198103 ratings of 412 movies made by 95325 users. These two datasets are all from real applications, and they are usually used to evaluate the computing performance of recommender system in social networks. Here, these datasets contain many features. For example, we could obtain some features of users from dataset MovieLens-100k, such as age, sex, job, hobby, and many others. More descriptions of these two datasets are shown in Table 1.

Table 1: Some Information on Two Datasets

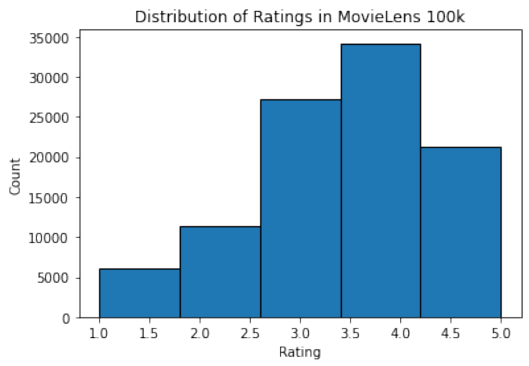
Datasets	# of users	# of movies	# of ratings	Sparsity
MovieLens-100k	943	1682	100000	6.3%
Netflix-6.2M	95325	412	6198103	15%

Figure 3 shows the rating distribution for each dataset; we observe all of them tend to be biased towards positive ratings. Rating scale of Movilens dataset is [1,5] and although Netflix dataset is [1,10], we moved it to [1,5] to make it easier to compare the results. We can construct an interaction matrix of size n×m , where n and m are the number of users and the number of items respectively. This dataset only records the existing ratings, so we can also call it rating matrix and we will use interaction matrix and rating matrix interchangeably in case that the values of this matrix represent exact ratings. Most of the values in the rating matrix are unknown as users have not rated the majority of movies. We also show the sparsity of this dataset. The sparsity is defined as 1 - number of nonzero entries / (number of users * number of items). Clearly, from Table 1, the interaction matrix is extremely sparse (i.e., Movielens sparsity = 93.695% & Netflix sparsity = 85%). Real world datasets may suffer from a greater extent of sparsity and has been a long-standing challenge in building recommender systems. A viable solution is to use additional side information such as user/item features to alleviate the sparsity. From Figure 3, we observe that for both datasets, the rating distribution is skewed towards rating of 4 (left plot is for ML dataset and the right plot is for Netflix dataset).

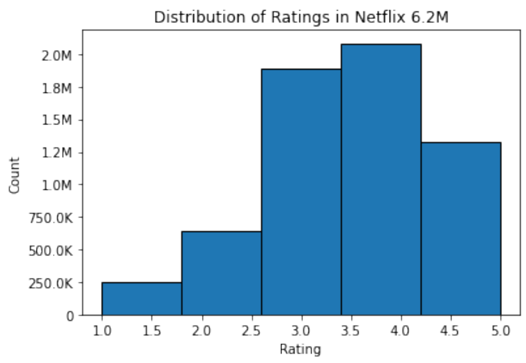
3.1.2 Dataset Cleaning

Data cleaning, also called data cleansing or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data [26]. In the real-world, data noise creates trouble for machine learning algorithms because if not trained properly, algorithms can think of noise to be a pattern and can start generalizing from it, which of course is undesirable.

MovieLens-100k: In the MovieLens dataset, ratings are very sparse and data points are mostly collected from very popular movies and highly engaged users. We wouldn't want movies that were rated by a small number of users because it's not credible enough. Similarly, users who have rated only a handful of movies should also not be taken into



(a) Movielens Dataset



(b) Netflix Dataset

Figure 3: Distribution of Ratings in Two Datasets

account. So, with all of that in mind, and some trial and error testing, we minimised the noise in the final movielens dataset by applying various filters a)To qualify a movie, a minimum of 10 users should have voted a movie, b) To qualify a user, a minimum of 50 movies should have voted by the user.Figure 4 shows the number of users who voted with our threshold of 10.After making the necessary modifications as per the threshold set, Figure 5 shows the number of votes by each user with our threshold of 50.

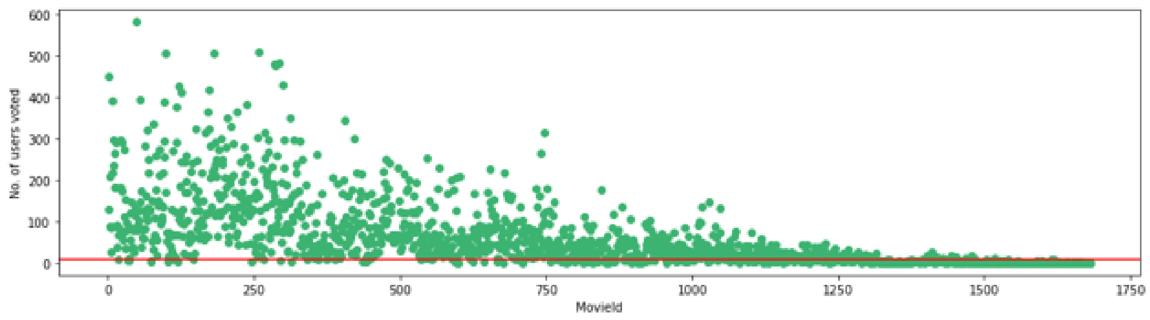


Figure 4: Visualization of the number of users who voted with our threshold of 10

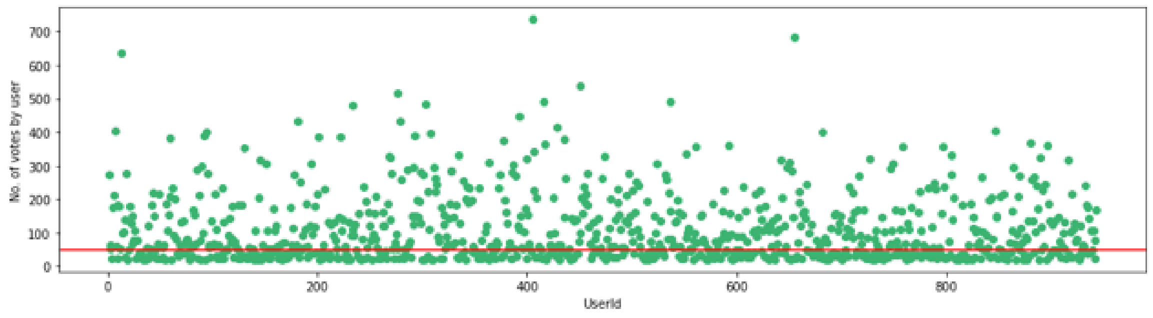
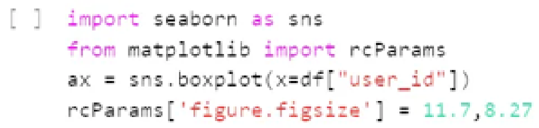


Figure 5: Visualization of the number of votes by each user with our threshold of 50

Netflix-6.2M: Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modeling and model skill in general can be improved by understanding and even removing these outlier values. In Netflix dataset, there are unusual values in a dataset which are problematic for many statistical analyses because they can cause tests to either miss significant findings or distort real results. To drop the outliers, we trimmed the data set, but replaced outliers with the nearest "good" data, as opposed to truncating them completely. This is known as Winsorization. As the dataset was huge, that is why we have applied Gaussian distribution for selecting more effective data range in the dataset (shown in Figure 6).



```
[ ]  import seaborn as sns
     from matplotlib import rcParams
     ax = sns.boxplot(x=df["user_id"])
     rcParams['figure.figsize'] = 11.7,8.27
```

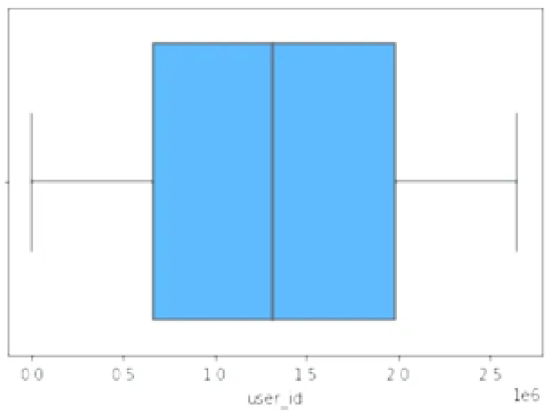


Figure 6: Removing Outliers using Standard Deviation :: Netflix-6.2M

3.1.3 Metrics

Here, three measures are employed to evaluate prediction accuracy. The first is to measure the accuracy of rating predictions, the second is to measure the accuracy of usage predictions, and the third is to measure how far from the regression line data points are; RMSE is a measure of how spread out these residuals are. MAE is defined by:

$$MAE = \frac{1}{N}\sum_{i=1}|y_i - \hat{y}| \tag{14}$$

Here, a testing dataset T is available in which the true rating r_{ui} is known, and the recommendation algorithms need to predict ratings $\hat{r}_{ui}$ for items in testing dataset. MSE is given by:

$$MSE = \frac{1}{N}\sum_{i=1}(y_i - \hat{y})^2 \tag{15}$$

And RMSE is defined by:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}(y_i - \hat{y}^2)} \tag{16}$$

Where, $\hat{y}$ is predicted value of y.

3.2 Movie Recommender Systems

1. MovieLens-100k: Making recommendation system using MovieLens Dataset, we followed two appraches, one is KNN Classification and another is ALS algorithm. Movielens 100k dataset is taken from Grouplens.org. We have used KNN Clssification to compute similarity with cosine distance metric which is very fast and more preferable than pearson coefficient. KNN is famous in a recommendation system for its faster predictive nature and low calculation time. Making movie recommender system using Pandas Dataframe for KNN Classification is very simple. We first check if the movie name input is in the database and if it is we use our recommendation system to find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie.

   Our second approach is Alternating Least Square algorithm. We have used ALS algorithm because implementation of ALS in Apache Spark ML is built for a larges-scale collaborative filtering problems. It is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets. To load the data as a spark dataframe, we import pyspark and instantiate a spark session. After removing noise from the dataset, we split the dataset into training and test sets into 80%, 20% ratio. Using ALS, we have found item-based and user-based movie recommender systems which is shown in section 4.

2. Netflix-6.2M: Netflix is a movie recommender system and virtual community website that recommends movies for its users to watch, based on their film preferences using collaborative filtering. Netflix 6.2M datatset is taken from the Kaggle website, which customizes user recommendation based on the ratings given by the user. We load the data as a spark dataframe and import pyspark and instantiate a spark session. Before moving into recommendations, We split our data into 2 parts train and test, train data (.80 & .20) is used to train our model based on certain algorithm and perform prediction on test data. Here, we approach two techniques to make recommender system, one is Linear Regression Analysis and another is ALS algorithm. We have used Linear Regression because it is most widely used predictive modelling. The main objective of the linear regression analysis is to check whether independent variable explains the dependent variable. Same as MovieLens, for Netflix, we applied ALS Method for making collaborative filtering algorithm (results shown in section 4).

4 Experimental Results and Analysis

4.1 Complexity Analysis

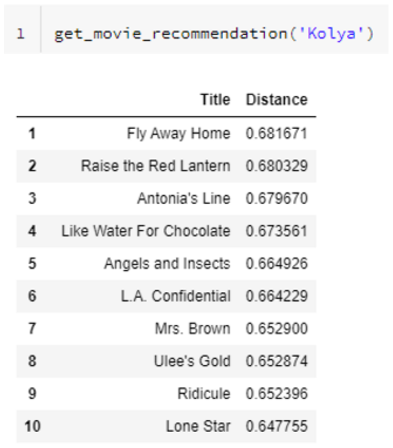
Complexity analysis is a method for determining an algorithm's execution time irrespective of the machine, the language, and the compiler.The performance of neighborhood-based recommendation method depends mostly on the similarity computation, since we should compute the similarity between two users in user-based CF algorithm or two items in item-based CF algorithm. Generally, the space and time complexity of user-based and item-based methods are shown in Table 2, where M denotes the number of users, N denotes the number of items, p denotes the maximum number of ratings per user, q denotes the maximum member of ratings per item, and k denotes the maximum number of neighbors [27]. Although the training time complexity is still $O(M^2p)$ or $O(N^2q)$, the time of computation is of great difference.

Table 2: The Space and Time Complexity of User-Based and Item-Based Methods

Method	Space	Training Time	Online Computing Time
user-based	$O(M^2)$	$O(M^2p)$	$O(Mk)$
item-based	$O(N^2)$	$O(N^2q)$	$O(Nk)$

4.2 k-Nearest Neighbour (kNN)

With kNN recommender systems, we tried to find similar movies and sort them based on their similarity distance. Figure 7 shows the output of the top 10 movies with their distances from the input movie. From the results, we assume that the top 10 recommended movies distance are so close and producing a promising output for the input movie, 'Kolya'. All the movies at the top are closest movie which are ideal for 'Kolya' movie.



```
get_movie_recommendation('Kolya')
```

	Title	Distance
1	Fly Away Home	0.681671
2	Raise the Red Lantern	0.680329
3	Antonia's Line	0.679670
4	Like Water For Chocolate	0.673561
5	Angels and Insects	0.664926
6	L.A. Confidential	0.664229
7	Mrs. Brown	0.652900
8	Ulee's Gold	0.652874
9	Ridicule	0.652396
10	Lone Star	0.647755

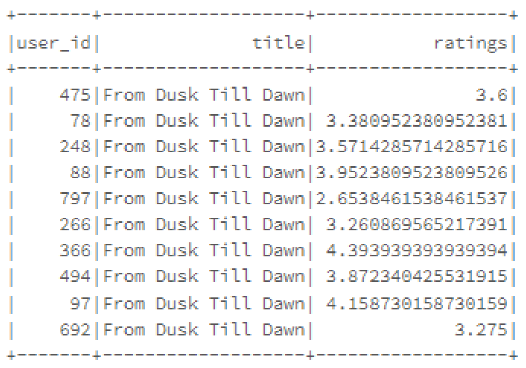
Figure 7: Movie Recommender System using KNN Algorithm :: MovieLens-100k

4.3 Alternating Least Square (ALS) Matrix Factorization

ALS Model is a powerful tool in building the recommendation system. Apache Spark provides a convenient API in building the model, however, most of time the model is not good enough at handling problems like data sparsity, cold start . . . etc. We need to combine with some strategies and user behavior analysis. We can also take a quick look on our recommendations. Figure 8 results user-based and item-based recommender system a) list of users for the movie id 17 and b) list of movies for the user id 6 for MovieLens-100k dataset. Along with this, Figure 9 demonstrates user and item-based movie recommender system for Netflix-6.2M dataset, where Figure 9(a) shows item-based and (b) user-based movie recommender system.

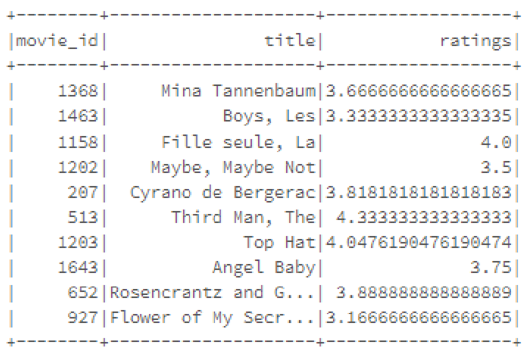
4.4 Linear Regression Approach

By default, the LinearRegression model assumes the name of the featuresCol and ratingsCol are 'features' and 'ratings', respectively. As long as the features are correctly formatted



```
+-------+-------------------+------------------+
|user_id|              title|           ratings|
+-------+-------------------+------------------+
|    475|From Dusk Till Dawn|               3.6|
|     78|From Dusk Till Dawn| 3.380952380952381|
|    248|From Dusk Till Dawn|3.5714285714285716|
|     88|From Dusk Till Dawn|3.9523809523809526|
|    797|From Dusk Till Dawn|2.6538461538461537|
|    266|From Dusk Till Dawn| 3.260869565217391|
|    366|From Dusk Till Dawn| 4.393939393939394|
|    494|From Dusk Till Dawn| 3.872340425531915|
|     97|From Dusk Till Dawn| 4.158730158730159|
|    692|From Dusk Till Dawn|             3.275|
+-------+-------------------+------------------+
```

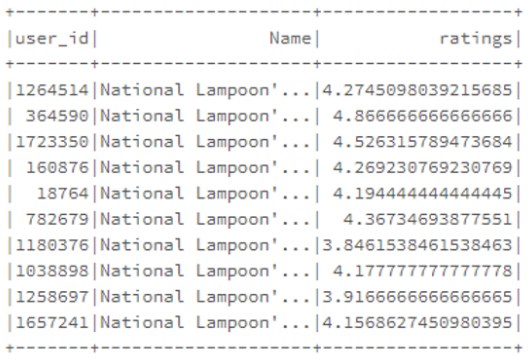
(a) Selecting the list of users for the movie which id is 17



```
+--------+--------------------+------------------+
|movie_id|               title|           ratings|
+--------+--------------------+------------------+
|    1368|     Mina Tannenbaum|3.6666666666666665|
|    1463|           Boys, Les|3.3333333333333335|
|    1158|     Fille seule, La|               4.0|
|    1202|    Maybe, Maybe Not|               3.5|
|     207|  Cyrano de Bergerac|3.8181818181818183|
|     513|      Third Man, The| 4.333333333333333|
|    1203|             Top Hat|4.0476190476190474|
|    1643|          Angel Baby|              3.75|
|     652|Rosencrantz and G...| 3.888888888888889|
|     927|Flower of My Secr...|3.1666666666666665|
+--------+--------------------+------------------+
```

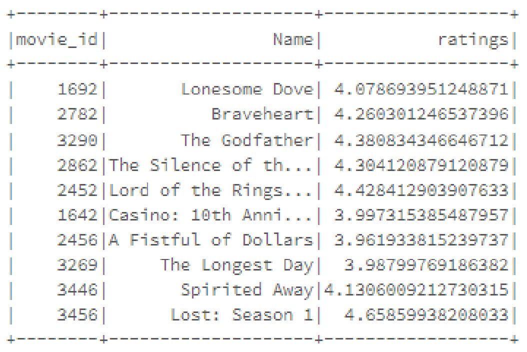
(b) Selecting the list of movies for the user whose id is 6

Figure 8: Movie Recommender System using ALS Algorithm :: MovieLens-100k



```
+-------+--------------------+------------------+
|user_id|                Name|           ratings|
+-------+--------------------+------------------+
|1264514|National Lampoon'...|4.2745098039215685|
| 364590|National Lampoon'...| 4.866666666666666|
|1723350|National Lampoon'...| 4.526315789473684|
| 160876|National Lampoon'...| 4.269230769230769|
|  18764|National Lampoon'...| 4.194444444444445|
| 782679|National Lampoon'...|  4.36734693877551|
|1180376|National Lampoon'...|3.8461538461538463|
|1038898|National Lampoon'...| 4.177777777777778|
|1258697|National Lampoon'...|3.9166666666666665|
|1657241|National Lampoon'...|4.1568627450980395|
+-------+--------------------+------------------+
```

(a) Selecting the list of users for the movie which id is 17



```
+--------+--------------------+------------------+
|movie_id|                Name|           ratings|
+--------+--------------------+------------------+
|    1692|       Lonesome Dove| 4.078693951248871|
|    2782|          Braveheart| 4.260301246537396|
|    3290|       The Godfather| 4.380834346646712|
|    2862|The Silence of th...| 4.304120879120879|
|    2452|Lord of the Rings...| 4.428412903907633|
|    1642|Casino: 10th Anni...| 3.997315385487957|
|    2456|A Fistful of Dollars| 3.961933815239737|
|    3269|     The Longest Day|  3.98799769186382|
|    3446|       Spirited Away|4.1306009212730315|
|    3456|      Lost: Season 1|  4.65859938208033|
+--------+--------------------+------------------+
```

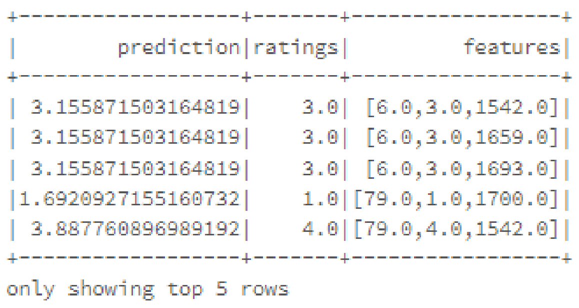
(b) Selecting the list of movies for the user whose id is 6

Figure 9: Movie Recommender System using ALS Algorithm :: Netflix-6.2M

into one column and the label column also exists, it doesn't matter what their name is. We imported LinearRegression from ML package of PySpark and would use as our algorithm for predictive modeling to create model. Using fit() method we passed train data to train our model. Based on this data supplied our model will predict the values for test data which we supplied later and got the result shown in Figure 10. If we stretch the output a little bit, we observe that 'ratings' and 'prediction' columns are close enough to be correct. As a result, we can conclude that our model prediction returns a promising output.

4.5 Measuring Accuracy

1. Alternating Least Square Method: We employed ALS method into two datasets, i.e., Movielens-100k & Netflix-6.2M. Figure 11 demonstrates the comparative analysis of accuracy level for the two datasets. Firstly, for Netflix-6.2M dataset, our model resulted in 0.92 RMSE, 0.85 MSE & 0.92 MAE. On the same node, for Movielens-100k dataset, our ALS method scored 1.053 RMSE, 1.109 MSE & 0.80 MAE, which is also considered a good-to-go model. An RMSE value of less than 2 is considered good. That said, this model can be further enhanced by adding features that would be recommended based on the top picks dependent on location or genre. We can see



```
+------------------+-------+-----------------+
|        prediction|ratings|         features|
+------------------+-------+-----------------+
| 3.155871503164819|    3.0| [6.0,3.0,1542.0]|
| 3.155871503164819|    3.0| [6.0,3.0,1659.0]|
| 3.155871503164819|    3.0| [6.0,3.0,1693.0]|
|1.692092715516073 2|    1.0|[79.0,1.0,1700.0]|
| 3.887760896989192|    4.0|[79.0,4.0,1542.0]|
+------------------+-------+-----------------+
only showing top 5 rows
```

Figure 10: Movie Recommender System using Linear Regression :: Netflix-6.2M

that all the values are near to zero or one. There is no correct value for MSE or MAE. Simply saying, the lower the value the better and 0 means the model is perfect. On that node, are model is doing pretty good on that.

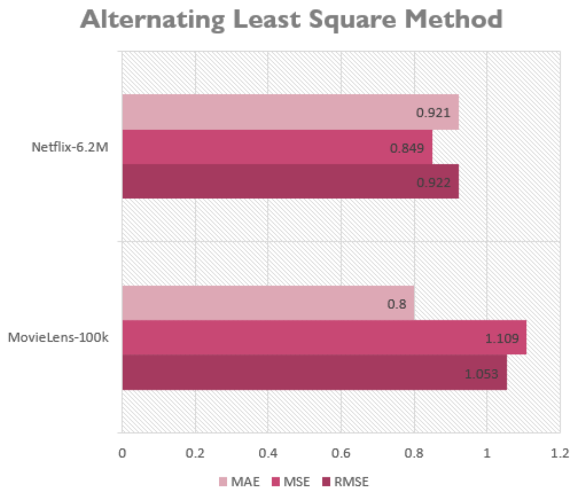


Figure 11: Comparison Between Netflix-6.2M & ML-100K Datasets ALS Method

2. Linear Regression Analysis: To verify how good our model is predicting the values of label and whether selecting linear regression as our algorithm for our model was good choice or not (means whether independent variable explains dependent variable) we use a evaluator. We applied Linear Regression Analysis for Netflix-6.2M dataset. RegressionEvaluator from ML package evaluates regression model. The metric used is R^2. We can use other metrics like rmse, mse, mae (root-mean square error, mean square error and mean absolute error) etc. From Figure 12, we assume that $R^2 = 0.93$. The R^2 value depends upon train and test data which is divided randomly, so the result might vary. R^2 value $\tilde{1}$ is a good fit. Hence we can conclude that our assumption that "Movie Rating Prediction" explains "Rating Prediction" is correct.

4.6 Ganglia UI Cluster Report

Ganglia is a scalable, distributed monitoring tool for high-performance computing systems, clusters and networks. The software is used to view either live or recorded statistics covering

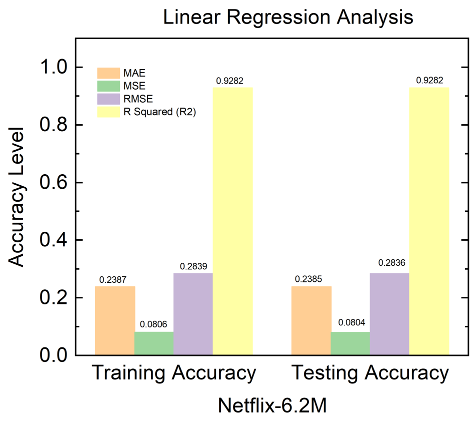


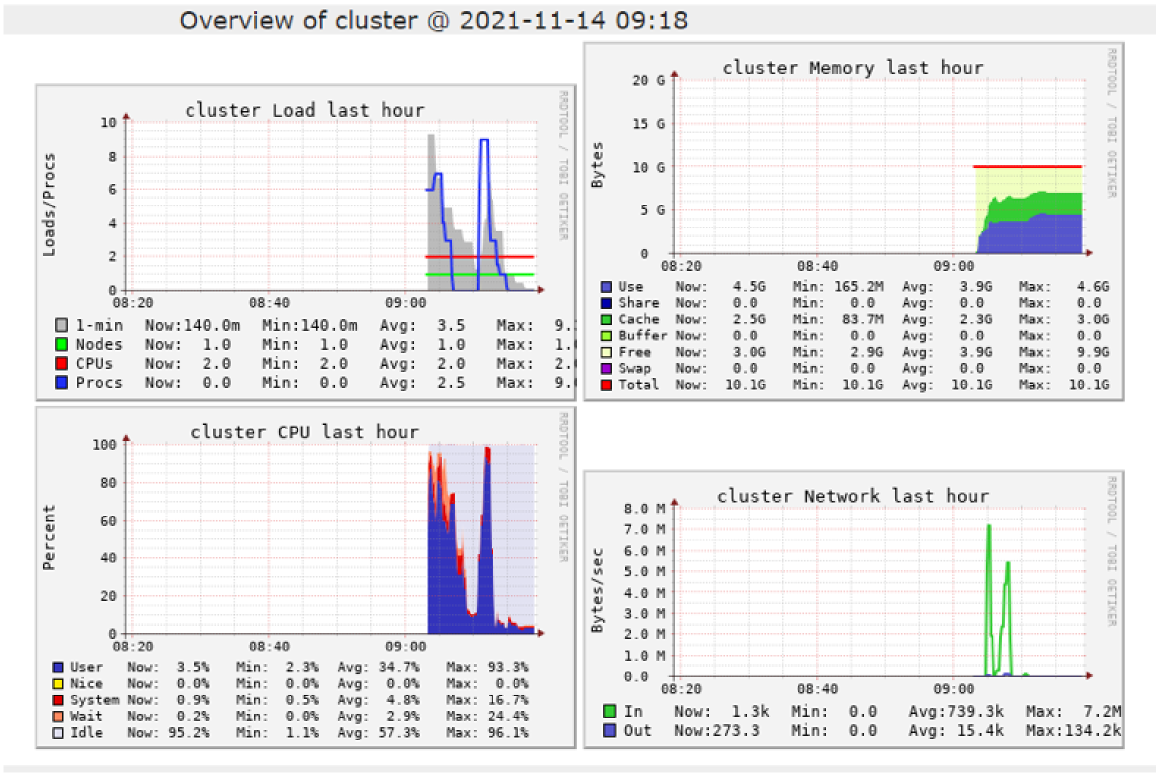
Figure 12: Accuracy Using Linear Regression Analysis :: Netflix-6.2M

metrics such as CPU load averages or network utilization for many nodes.

1. MovieLens-100k :In the case below (shown in Figure 13), we can see a Ganglia dashboard showing us that the cluster configuration for MovieLens-100k dataset is bigger than the necessary, by analyzing the low use of memory, CPU and a balanced server load distribution. Meanwhile, a single cluster configuration is not always a golden rule: for some DAGs it may work perfectly fine, but for others the same configuration might not be enough, causing a deterioration in performance and even not being able to completely load data. On this dash is important to note on the "cluster Memory last hour" report the "Use" graph, in order to understand how much memory was allocated for this DAG run. On the "cluster CPU last hour" dash, we can look at "User" and understand how much of the CPU is being used, also in "Idle" we can see how inactive the CPU has gone through.

   From this dashboard (shown in Figure 13), we can notice that the CPU has a maximum use of almost 93.3% of its capacity and having an average idle time of almost 57.3%. About the memory, we can see that the maximum usage was only 4.6 GB, almost half of the full capacity. Also, the two nodes — the squares on "Server Load Distribution" — are getting balanced, and that is good. It is important to keep in mind that the memory allocated by the cluster — in the Ganglia dashboard — to the program's pre-allocation, does not directly reflect the amount of data allocated. There are other operations that go to Memory, in addition to calculating the cluster on the amount of memory that is already allocated in relation to the size of the machine (in order to avoid memory grant operations with the OS). After executing the whole DAG, the total run time on our Airflow pipeline with the default cluster was about 12 minutes 15 seconds.

2. Netflix-6.2M: Running again the DAG with this new cluster for Netflix-6.2M, we can find this Ganglia dashboard, as shown in Figure 14. On the "cluster Memory last hour" dash, we can notice that the maximum usage of memory was 5.7 GB and about the "cluster CPU last hour" dashboard, we have a maximum usage of CPU of 98.9% and an average idle time of 64.2%. Also, in this case, we opted for the auto-scaling configuration, leaving to the cluster the option of using up to three nodes, if necessary. In the picture shown in Figure 14, only two are being used, and they are



```
1-min   Now:140.0m   Min:140.0m   Avg:  3.5   Max:  9.
Nodes   Now:  1.0    Min:  1.0    Avg:  1.0   Max:  1.
CPUs    Now:  2.0    Min:  2.0    Avg:  2.0   Max:  2.
Procs   Now:  0.0    Min:  0.0    Avg:  2.5   Max:  9.
```

```
Use    Now:  4.5G  Min: 165.2M  Avg:  3.9G  Max:  4.6G
Share  Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Cache  Now:  2.5G  Min:  83.7M  Avg:  2.3G  Max:  3.0G
Buffer Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Free   Now:  3.0G  Min:   2.9G  Avg:  3.9G  Max:  9.9G
Swap   Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Total  Now: 10.1G  Min:  10.1G  Avg: 10.1G  Max: 10.1G
```

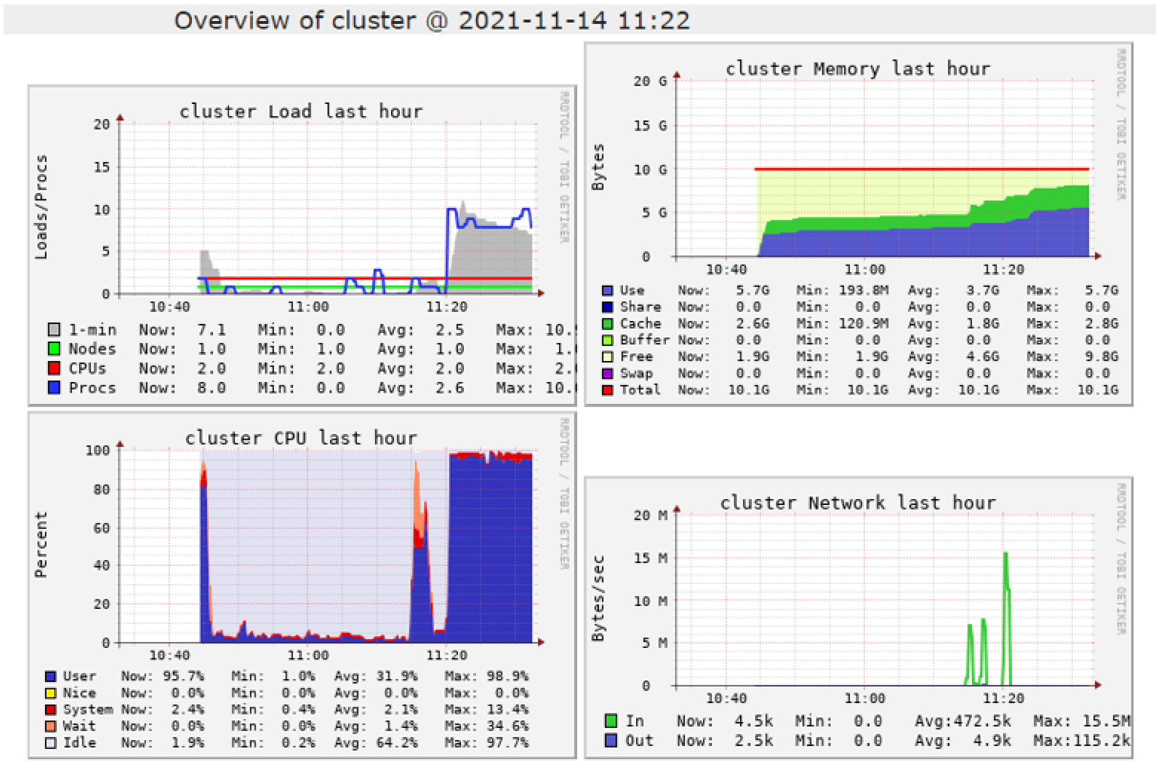
```
User   Now:  3.5%  Min:  2.3%  Avg: 34.7%  Max: 93.3%
Nice   Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
System Now:  0.9%  Min:  0.5%  Avg:  4.8%  Max: 16.7%
Wait   Now:  0.2%  Min:  0.0%  Avg:  2.9%  Max: 24.4%
Idle   Now: 95.2%  Min:  1.1%  Avg: 57.3%  Max: 96.1%
```

```
In   Now:  1.3k  Min:  0.0   Avg:739.3k  Max:  7.2M
Out  Now:273.3   Min:  0.0   Avg: 15.4k  Max:134.2k
```

Figure 13: Ganglia UI Cluster Report :: Movielens Dataset

getting balanced as well. After executing the whole DAG in our Airflow structure with this minimum resources cluster, the total run time was about 18 minutes.

3. Default and Minimum Resources Clusters Configurations - usage Overall After running our DAG example with both different types of clusters, we are able to compare its results (shown in Figure 15). Talking about the memory usage, when using the MovieLens configuration, we were underusing its capacity, having only 4.6 GB, whereas Netflix used 5.7 GB of the total memory of 15.3 GB. One good point to notice here is the difference in memory usage when having less memory available: probably, it looks to be related to the buffer area — a bigger machine would reserve a bigger area for memory or proportional to the allocated area. Looking at the maximum CPU usage that happened, with the MovieLens configuration, we have its maximum at around 93.3%, which looks okay. But when we changed to the Netflix resources configuration, we reached around 98.9%, once again doing a better use. About the CPU average idle time, with the MovieLens configuration, we have $\tilde{5}7\%$ of total runtime. Looking at the Netflix configuration, we have 64.2%, which tells us that we are making better use of our CPU, having a good time of average idle. And for last, about our total runtime/hour of our DAG, using default configuration we have $\tilde{1}2$ minutes, 20% of an hour. Using the Netflix configuration, we have 18 minutes, 30% of an hour. Here, increasing the total runtime is not a good thing, but as we can see, it increased by only 6 minutes and for us, it still works fine. When thinking about the whole scenario, we can see that our Movielens resources configuration is doing a better use of all that is available, despite the fact that it has increased the total runtime by 10



```
1-min  Now:  7.1   Min:  0.0   Avg:  2.5   Max:  10.
Nodes  Now:  1.0   Min:  1.0   Avg:  1.0   Max:  1.
CPUs   Now:  2.0   Min:  2.0   Avg:  2.0   Max:  2.
Procs  Now:  8.0   Min:  0.0   Avg:  2.6   Max:  10.
```

```
Use    Now:  5.7G  Min: 193.8M  Avg:  3.7G  Max:  5.7G
Share  Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Cache  Now:  2.6G  Min: 120.9M  Avg:  1.8G  Max:  2.8G
Buffer Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Free   Now:  1.9G  Min:   1.9G  Avg:  4.6G  Max:  9.8G
Swap   Now:  0.0   Min:   0.0   Avg:  0.0   Max:  0.0
Total  Now: 10.1G  Min:  10.1G  Avg: 10.1G  Max: 10.1G
```

```
User   Now: 95.7%  Min:  1.0%  Avg: 31.9%  Max: 98.9%
Nice   Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
System Now:  2.4%  Min:  0.4%  Avg:  2.1%  Max: 13.4%
Wait   Now:  0.0%  Min:  0.0%  Avg:  1.4%  Max: 34.6%
Idle   Now:  1.9%  Min:  0.2%  Avg: 64.2%  Max: 97.7%
```

```
In   Now: 4.5k  Min: 0.0  Avg:472.5k  Max: 15.5M
Out  Now: 2.5k  Min: 0.0  Avg:  4.9k  Max:115.2k
```

Figure 14: Ganglia UI Cluster Report :: Netflix Dataset

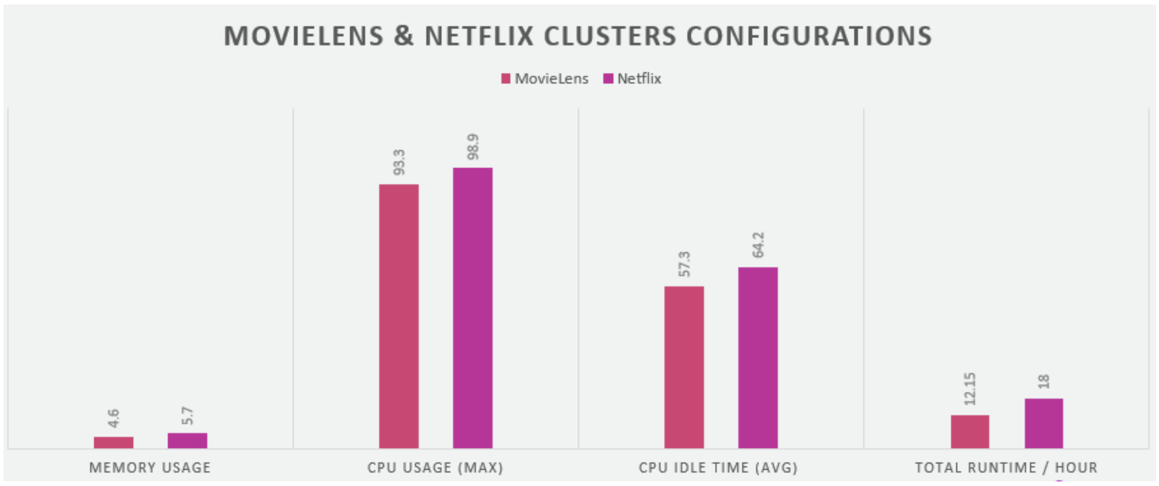


Figure 15: MovieLens and Netflix Clusters Configurations - usage Overall

5 Conclusions

Apache spark is well suited for applications which require high speed query of data, transformation and analytics results. Therefore, the recommendation system developed in this research is implemented on Apache Spark. Also, the matrix factorization using Alternating Least Squares (ALS) algorithm which is a type of collaborative filtering is used to solve overfitting issues in sparse data and increases prediction ac-curacy. The overfitting problem arises in the data as the user-item rating matrix is sparse. In this research a recommendation system for Movielens and Netflix using alternating least squares (ALS) matrix factorization method, kNN Classification & Linear Regression Analysis on Apache Spark MLlib is developed. From ALS algorithm, we recognized that item-based filtering is more convenient than user-based. The research shows that the accuracy for MovieLens-100k & Netflix-6.2M are 1.053 RMSE, 1.109 MSE & 0.80 MAE, and 0.92 RMSE, 0.85 MSE & 0.92 MAE using ALS matrix factorization method respectively. The accuracy for RMSE, MSE & MAE for both datasets are considered a good-to-go model. An RMSE value of less than 2 is considered good. That said, this model can be further enhanced by adding features that would be recommended based on the top picks dependent on location or genre. The approach used is highly scalable, and can be used with computational clusters using HDFS for much larger data files. By using a parallel architecture we can make better use of hardware instead of using a pythonic serial calculation approach. This reduces runtimes for larger calculations. We have improved upon the fellow research paper which mentioned about recommender system using Correntropy similarity that used collaborative-filtering. This could be a combination of having more training data available and the Matrix Factorization approach used. In conclusion, our model works quite well- a movie recommendation system based on user behavior.

Comparing the results achieved by the various algorithms, we can see that the performance of recommender systems is highly dependent on various characteristics of the dataset such as the number of items, the number of users, its sparsity, and the behavioral variability of the various users in terms of the items they buy/see.Hence, due to the limited machine configuration crisis, we struggled to employ the Spark Dataframe into our local machine. Apart from that, using the alternative cloud based platform such as Microsoft Azure Databricks (Student Edition) hinders to create multiple Master / Slave nodes under a single cluster. However, remaining with default cluster settings in Databricks Community Edition, brings a minimal cluster settings i.e., 8-core CPU with 2 nodes.In a parallel note, MovieLens-100k and Netflix-6.2M dataset creates much hardle to run into the default cluster settings. For improving the accuracy and interpretability of our collaborative filtering algorithm, we will consider the following improvements:

a Hybrid recommender systems: the major weakness of collaborative filtering is that by forgoing the actual characteristics of items (for movies, meta-information such as genres, actor/actresses, director, country of origin) hurts both recommender accuracy and interpretability of recommendations. In practice, industrial recommender systems use hybrid approaches that combine both user similarity (collaborative filtering) and item characteristics (content-based approach). There exists a Python implementation of such a hybrid approach known as LightFM which merits exploration.

b Reducing dataset size and user-item segmentation: one reason why our collaborative filtering model struggled to generate sound recommendations for all users is that our model training included every user (e.g. single rating users, users with few ratings or many ratings). A more ideal strategy would be to segment our ratings data into seperate but homogeneous user datasets and train different recommender systems on those separate data chunks, potentially improving our results.

In future, the recommendation system will be integrated with content based filtering, users click stream predictive analytics, deep collaborative filtering, and convolutional neural network based [28–31] recommendation system. The integration might use results produced by these different systems and correlate these results to show recommendations to the online users. Thus, the future research direction will be focusing on hybrid models of recommendation systems [32–39].

References

[1] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: Introduction and challenges," *Recommender Systems Handbook*, p. 1–34, 2015.

[2] A. Werner-Stark and Z. Nagy, "A heuristic method to recommendation systems," *2020 6th International Conference on Information Management (ICIM)*, p. 200–204, 2020.

[3] A. Pal, P. Parhi, and M. Aggarwal, "An improved content based collaborative filtering algorithm for movie recommendations," *2017 Tenth International Conference on Contemporary Computing (IC3)*, p. 1–3, 2017.

[4] R. Ji, Y. Tian, and M. Ma, "Collaborative filtering recommendation algorithm based on user characteristics," *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*, p. 56–60, 2020.

[5] J. Sun, Z. Wang, X. Luo, P. Shi, W. Wang, L. Wang, J.-H. Wang, and W. Zhao, "A parallel recommender system using a collaborative filtering algorithm with correntropy for social networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, p. 91–103, 2020.

[6] D. Lalwani, D. V. L. N. Somayajulu, and P. R. Krishna, "A community driven social recommendation system," *2015 IEEE International Conference on Big Data (Big Data)*, 2015.

[7] K. B. Fard, M. Nilashi, and N. Salim, "Recommender system based on semantic similarity," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 3, no. 6, 2013.

[8] M. Deshpande and G. Karypis, "Item-based top- n recommendation algorithms," *ACM Transactions on Information Systems*, vol. 22, no. 1, p. 143–177, 2004.

[9] P. Ahlgren, B. Jarneving, and R. Rousseau, "Requirements for a cocitation similarity measure, with special reference to pearsons correlation coefficient," *Journal of the American Society for Information Science and Technology*, vol. 54, no. 6, p. 550–560, Apr 2003.

[10] J. Bobadilla, A. Hernando, F. Ortega, and A. Gutiérrez, "Collaborative filtering based on significances," *Information Sciences*, vol. 185, no. 1, p. 1–17, 2012.

[11] X. Wu, Y. Huang, and S. Wang, "A new similarity computation method in collaborative filtering based recommendation system," *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, 2017.

[12] J. Herlocker, J. A. Konstan, and J. Riedl, "An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms." [Online]. Available: https://link.springer.com/article/10.1023/A:1020443909834citeas

[13] Pero and T. Horváth, “Opinion-driven matrix factorization for rating prediction,” *User Modeling, Adaptation, and Personalization Lecture Notes in Computer Science*, p. 1–13, 2013.

[14] P. Kumar, V. Kumar, and R. S. Thakur, “A new approach for rating prediction system using collaborative filtering,” *Iran Journal of Computer Science*, vol. 2, no. 2, p. 81–87, 2018.

[15] A. Bonfietti and M. Lombardi, “The weighted average constraint,” vol. 7514, 10 2012, pp. 191–206.

[16] M. Hofer, “Mean centering,” *The International Encyclopedia of Communication Research Methods*, p. 1–3, 2017.

[17] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” Jan 1998. [Online]. Available: https://arxiv.org/abs/1301.7363

[18] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An algorithmic framework for performing collaborative filtering,” *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR 99*, 1999.

[19] K. Molugaram and G. S. Rao, “Random variables,” *Statistical Techniques for Transportation Engineering*, p. 113–279, 2017.

[20] W. Wang, H. Zhao, and X. Zeng, “Geometric algebra correntropy: Definition and application to robust adaptive filtering,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 6, p. 1164–1168, 2020.

[21] W. Liu, P. P. Pokharel, and J. C. Principe, “Correntropy: Properties and applications in non-gaussian signal processing,” *IEEE Transactions on Signal Processing*, vol. 55, no. 11, p. 5286–5298, 2007.

[22] B. Chen, X. Liu, H. Zhao, and J. C. Principe, “Maximum correntropy kalman filter,” *Automatica*, vol. 76, p. 70–77, 2017.

[23] X. Luo, J. Deng, W. Wang, J.-H. Wang, and W. Zhao, “A quantized kernel learning algorithm using a minimum kernel risk-sensitive loss criterion and bilateral gradient technique,” *Entropy*, vol. 19, no. 7, p. 365, 2017.

[24] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita, “Big data analytics in the cloud: Spark on hadoop vs mpi/openmp on beowulf,” *Procedia Computer Science*, vol. 53, p. 121–130, 2015.

[25] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, and et al., “Apache spark,” *Communications of the ACM*, vol. 59, no. 11, p. 56–65, 2016.

[26] E. Rahm and H. H. Do, “Data cleaning: Problems and current approaches,” *IEEE Data Eng. Bull.*, vol. 23, pp. 3–13, 2000.

[27] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods." in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 107–144. [Online]. Available: http://dblp.uni-trier.de/db/reference/rsh/rsh2011.htmlDesrosiersK11

[28] R. R. Chowdhury, M. S. Hossain, R. U. Islam, K. Andersson, and S. Hossain, "Bangla handwritten character recognition using convolutional neural network with data augmentation," *2019 Joint 8th International Conference on Informatics, Electronics Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, 2019.

[29]

[30] "Facial expression recognition using convolutional neural network with data augmentation." [Online]. Available: https://ieeexplore.ieee.org/document/8858529

[31] M. Z. Islam, M. S. Hossain, R. U. Islam, and K. Andersson, "Static hand gesture recognition using convolutional neural network with data augmentation," *2019 Joint 8th International Conference on Informatics, Electronics Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, 2019.

[32] S. Sinitsyn, V. Panchenko, V. Kharchenko, A. Kovalev, and P. Vasant, "The concept of information modeling in interactive intelligence systems," *Advances in Intelligent Systems and Computing Intelligent Computing and Optimization*, p. 249–259, 2019.

[33] K. M. Alhendawi and A. A. Al-Janabi, "An intelligent expert system for management information system failure diagnosis," *Intelligent Computing Optimization Advances in Intelligent Systems and Computing*, p. 257–266, 2018.

[34] M. Biswas, S. U. Chowdhury, N. Nahar, M. S. Hossain, and K. Andersson, "A belief rule base expert system for staging non-small cell lung cancer under uncertainty," *2019 IEEE International Conference on Biomedical Engineering, Computer and Information Technology for Health (BECITHCON)*, 2019.

[35] S. Kabir, R. U. Islam, M. S. Hossain, and K. Andersson, "An integrated approach of belief rule base and deep learning to predict air pollution," *Sensors*, vol. 20, no. 7, p. 1956, 2020.

[36] A. A. Monrat, R. U. Islam, M. S. Hossain, and K. Andersson, "A belief rule based flood risk assessment expert system using real time sensor data streaming," *2018 IEEE 43rd Conference on Local Computer Networks Workshops (LCN Workshops)*, 2018.

[37] R. Karim, M. S. Hossain, M. S. Khalid, R. Mustafa, and T. A. Bhuiyan, "A belief rule-based expert system to assess bronchiolitis suspicion from signs and symptoms under uncertainty," *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016 Lecture Notes in Networks and Systems*, p. 331–343, 2017.

[38] M. S. Hossain, A. A. Monrat, M. Hasan, R. Karim, T. A. Bhuiyan, and M. S. Khalid, "A belief rule-based expert system to assess mental disorder under uncertainty," *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, 2016.

[39] M. S. Hossain, I. B. Habib, and K. Andersson, "A belief rule based expert system to diagnose dengue fever under uncertainty," *2017 Computing Conference*, 2017.