Chapter 15
Recommender System in the Context of Big Data: Implementing SVD-Based Recommender System using Apache Hadoop and Spark

Khadija Ateya Almohsen
Ahlia University, Bahrain

Huda Kadhim Al-Jobori
Ahlia University, Bahrain

ABSTRACT

The increasing usage of e-commerce website has led to the emergence of Recommender System (RS) with the aim of personalizing the web content for each user. One of the successful techniques of RSs is Collaborative Filtering (CF) which makes recommendations for users based on what other like-mind users had preferred. However, as the world enter Big Data era, CF has faced some challenges such as: scalability, sparsity and cold start. Thus, new approaches that overcome the existing problems have been studied such as Singular Value Decomposition (SVD). This chapter surveys the literature of RSs, reviews the current state of RSs with the main concerns surrounding them due to Big Data, investigates thoroughly SVD and provides an implementation to it using Apache Hadoop and Spark. This is intended to validate the applicability of, existing contributions to the field of, SVD-based RSs as well as validated the effectiveness of Hadoop and spark in developing large-scale systems. The results proved the scalability of SVD-based RS and its applicability to Big Data.

INTRODUCTION

Advances in technology, the wide spread of its usage and the connectivity of everything to the Internet have led the world to experience unusual rate of generating and storing data; resulting in what is being called Big Data phenomenon. As a consequence of this emerging fluid of data, normal tasks and activities become challenges. For instance, browsing the web and searching for interesting information or products

DOI: 10.4018/978-1-5225-0182-4.ch015

is a routine and common task. However, the massive amount of data on the web is expanding the noise there making it harder and more time consuming to choose the interesting pieces of information from all this noise (Villa, 2012; Schelter & Owen, 2012).

Likewise, the currently available systems, technologies and tools show their limitation in processing and managing this massive amount of data. This leads to the invention of new technologies, such as Map Reduce of Google, Hadoop of Yahoo! And Spark from University of California, Berkely. These technologies are just like the telescopes which allow us to comprehend the universe (Schönberger & Cukier, 2013). With this in mind, existing systems have been adapted to meet Big Data by using the newly invented tools and technologies. One of these systems is recommender system and it is the one under study in this chapter.

Recommender systems have been implemented long time ago by several Internet giants; like Amazon.com, Facebook and Google. These systems suggest new items that might be of interest to the user by analyzing user's profiles, their activities on the websites as well as their purchase history; if applicable. However, Big Data increases the cognitive load on the user, posing more challenges on recommender systems as it should provide many recommendations of high quality by analyzing huge data of customers and products. In another word, high quality, scalability and performance become concerns. (Berkeley School: Lecture12 -Analyzing Big Data with Twitter: Recommender Systems, 2012; Chiky, et al, 2012; Thangavel, et al, 2013). This encourages more research work on recommendation algorithms and the use of new tools and frameworks like Appache Hadoop and Spark, i.e. Big Data tools, in the development of scalable systems as well as preventing the computational cost from going up while processing vast amount of data (Schelter & Owen, 2012; Zhao & Shang, 2010).

This chapter provides a comprehensive and self-contained description of this research area. Such a work will constitute a milestone for studies on Big Data; since it will provide review of key references which will be useful in the search for research topics dealing with Big Data. To achieve such goal, this chapter will review the literature on Big Data and recommendation engines. In addition, survey the promising approaches of recommender systems that are expected to perform well while handling Big Data; such as Singular Value Decomposition. Furthermore, assess the applicability and viability of Big Data technologies (i.e. Apache Hadoop and Spark) to the field of recommendation system as well as develop a scalable recommender system that can handle large volume of data.

BACKGROUND

Recommender System Overview

Recommender system's (RS) main mission is to find the taste of a person and automatically suggests, new, relevant content for him/her. These suggestions aid to decision-making; for example: which item to buy, which music to listen to, or which news to read. This is achieved by finding patterns in people opinions, even though their opinions vary. These patterns are useful in predicting what a user might like or dislike. For illustration, people like something which is similar to what they liked before, or they like what others of similar taste and opinion seem to like (Owen et al., 2012; Ricci et al., 2011).

Most of recommender systems aim to provide a personalize websites by suggesting different items to different users. However, there are some recommendations which are non-personalized such as: Top

ten selections of movies in a magazine or a web site. Such recommendations are much easier than personalized one (Ricci et al., 2011)

In the case of personalized recommendations, the system comes up with a ranked list of most relevant items. This is achieved by analyzing the user preference that is usually expressed explicitly- in terms of rating- or expressed implicitly through user action and activities in the website; for example: by considering the navigation to a specific product page. After obtaining the ranked list of recommendation, the user can view the recommendations and have the choice of accepting or rejecting them. In addition, the user may have the option of providing a feedback on them (Ricci et al., 2011).

Recommender System's Problem Formulation

Suppose that a Big Data set records the preferences of big number of users; denoted by m; for some or all of n items. The preference record usually takes the form of tuple (userID, itemID, rating); where rating takes a value on a numerical scale (for example from 1-5) and that expresses how much the user holding userID likes the item with itemID.

Let R be a user–item matrix of size $m \times n$, where m is the number of users and n is number of items. This matrix represents the preference records such that each one of the m rows represents the preferences of one user to the available n items. In another word, the value of a particular cell R_{ij} either holds the rating given by user i to item j or null if the user did not rate the item yet, as shown in Figure 1. In most of the cases, this matrix is sparse because each user does not normally rate all the items in the data set.

The mission of a RS is to predict the missing ratings; i.e. predict how a user would rate an item in the future. This aids the recommender system in recommending items that are predicted to receive high rating by the user (Melville & Sindhwani, 2010).

Recommender System Approaches

The commonly used approach of recommender system is collaborative filtering. It makes recommendation based on the existing relationship between users and items (i.e. who bought which item? Who viewed which item? Who liked which item?). This approach is relying on the following assumptions:

Figure 1. User-item matrix

	$item_1$	$item_2$	$item_3$	...	$item_n$
$user_1$		5	2		1
$user_2$	3				
$user_3$	1		3		
.					
.					
.					
$user_{m-1}$	5		4		2
$user_m$		4			3

1. People may have similar taste and preferences.
2. People interest and preferences are stable.
3. We can rely on past preferences to predict their new choices.

The two common variations of collaborative filtering approach are:

1. **User-based collaborative filtering** (which is also called nearest-neighbor collaborative filtering):

It examines the entire data set of users and items to generate recommendations by identifying users that have similar interests to the target one and then recommends items that have been bought by others and not the target user. This proceeds by constructing user-item matrix, computing some statistical metrics on it to measure the similarity between different rows and finding the nearest neighbors. These neighbors are supposed to have similar interest with the target user. This will be followed by combining the neighbors' preferences and finding the top N items that have been rated highly by neighbors and not by the target user. These N items will form the top N recommendations. (Thangavel, *et al*, 2013).

Despite the fact that this approach has been adapted widely, it suffers from scalability problem which was not considered a big issue few decades ago when the number of users and items was relatively small. However, as the data set size increases in big data era, computing the similarity between users is increasing exponentially because of the need for comparing each user with all the other users. Moreover, as the users interact with more items and change their preferences, the similarity needs to be recomputed; i.e. similarity pre-computation becomes useless. This is degrading the performance of RSs and that is why it is being considered as a big problem today. Furthermore, having a sparse user-item matrix, which is usually the case because users interact with relatively small set of items, also adds to the difficulty of computing user's similarity since the number of common items is relatively small if not zero (Thangavel, et al, 2013; Walunj & Sadafale, 2013b; Owen et al., 2012; Lee & Chang, 2013).

2. Item-based collaborative filtering:

It examines the set of items rated by the target user and finds other items similar to them (which are called neighbors), by considering other users' preferences. With the hope of finding neighbors, each item will be represented by a vector of the ratings given by the different users, and then, the similarity of two items will be measured by computing the similarity between their vectors. These neighbors will form the recommendations and will be ranked after predicting the preference of the target user for each one of them. The prediction $P_{u,i}$ of the target user to one of the neighbors, item i, is given by:

$$P_{u,i} = \frac{\sum_{i=1}^{N} Sim(i,j) \times R_{u,j}}{\sum_{i=1}^{N} Sim(i,j)}$$

Where N is the number of neighbors, $Sim(i,j)$ is the similarity between the item j and its neighbor i, $R_{u,j}$ is the rating given by user u to item j. (Lee & Chang, 2013; Gong, et al, 2009).

However, measuring the similarities between items takes long time and consumes lots of computer resources. This is the main pitfall of this method. Anyhow, changes in items are not as frequent as changes in users and, thus, such computations can be pre-calculated in an offline mode. Another strength of this algorithm is that it is not affected by having a sparse user-item matrix. This is because with large number of user, there will be enough number of ratings for each item which enable measuring the similarity between the different items and getting significant statistics. (Thangavel, et al, 2013; Walunj & Sadafale, 2013b; Lee & Chang, 2013).

Generally speaking, CF whether it is an item-based or a user based approach, has a well-known strength in which it is not domain specific, and thus, does not rely on the items' properties and attributes. That is why it is applicable to different domains: movie recommendation, book recommendation, flowers, food and others. However, CF suffers from the following problems:

1. **Scalability**: RSs are being fed with massive amount of data which should be processed rapidly. However, CF algorithms computation time grows up with the continuous increase in the number of users and items (Lee & Chang, 2013).
2. **Data Sparsity**: In an e-commerce website, users usually rate small fraction of all the available items resulting in sparse data set. This degrades the accuracy of the RS because it complicates the process of finding similarities between users as the number of common items becomes relatively small (Lee & Chang, 2013).
3. **Cold-Start Problems**: This problem emerged as a consequence of data sparsity problem; where new users cannot get personalized recommendation unless they rate a sufficient number of items. Likewise, new items cannot be recommended before getting reasonable number or ratings (Kabore, 2012).
4. **Synonymy:** Different products have different names in the data set even if they are similar to each other. In this case, a standard CF RS will treat them differently and will not infer the hidden association between them. For illustration, "cartoon film" and "cartoon movie" are two phrases refereeing to the same item. However, ordinary implementations of CF algorithms had treated them differently! (Sarwar et al, 2000).
5. **Grey Sheep**: It addresses users whose opinions do not match with any other group of users. Consequently, CF cannot serve grey sheep since it mainly relies on the similarity between users' previous preferences (Walunj & Sadafale, 2013a).

The aforementioned, standard, implementation of item-based and user-based CF are following memory-based approach in which the entire data set is kept in memory while processing it and searching for similarities between users or items in order to make recommendation. The other approach of implementing CF algorithm is called model based approach in which the data set is used in an offline mode to generate a model by utilizing some data mining, machine learning or statistical techniques. This model could be used later on to predict the ratings for unseen items without the need of processing the entire data set again and again. Examples of this approach are: decision trees, clustering methods and matrix factorization models (Pagare & Patil, 2013).

Point often overlooked is that model-based approach generates predictions with lower accuracy when compared with memory based approach. However, it has better scalability. Thus, many researchers are investing their effort in studying and enhancing model-based CF algorithms. One of these algorithms is Singular value decomposition (SVD) based recommenders and it is the one under study in this chapter.

Singular Value Decomposition Recommender System:

SVD is one of the famous matrix factorization techniques that decompose a matrix R of size $m \times n$ and rank = r into three matrices U, S and Vas follows:

$$R = U.S.V^T$$

where:

U: an orthonormal matrix of size $m \times r$ holding left singular vectors of R in its columns; i.e. its r columns hold eigenvectors of the r nonzero eigenvalues of RR^T.
S: a diagonal matrix of size $r \times r$holding the singular values of R in its diagonal entries in decreasing order; i.e. $s_1 \geq s_2 \geq s_3 \geq \ldots\ldots \geq s_r$. These r values are the nonnegative square roots of eigenvalues of RR^T.
V: an orthonormal matrix of size $n \times r$ holding the right singular vectors of R in its columns; i.e. its r columns hold eigenvectors of the r nonzero eigenvalues of R^TR .

Furthermore, S could be reduced by taking the largest k singular values only and thus obtain S_k of size $k \times k$. Accordingly, U and V could be reduced by retaining the first k singular vectors and discarding the rest. In another word, U_k is generated by eliminating the last r-k column of U and, similarly, V_k is generated by eliminating the last r-k column of V. This will yield U_k of size $m \times k$ and V_k of size $n \times k$. As a consequence, $R_k = U_k \, . \, S_k \, . \, V^T_k$ and $R_k \approx R$, where R_k is the closest rank k approximation to R (Lee & Chang, 2013; Sarwar et al., 2000; Berry, et al, 1995).

Applying SVD on Recommender System

This approach assumes that the relationship between users and items as well as the similarity between users / items could be induced by some hidden lower dimensional structure in the data. For illustration, the ratings given by a specific user to a particular movie, assuming that items are movies, depends on some implicit factors like the preference of that user across different movie genres. As a matter of fact, it treats users and items as unknown feature vectors to be learnt by applying SVD to user–item matrix and breaking it down into three smaller matrices: U, V and S (Melville & Sindhwani, 2010).

With this in mind, applying SVD to RSs proceeds as follow:

Construct user-item matrix R of size $m \times n$ from the input data set; which is usually a sparse matrix. Unfortunately, this sparsity degrades the accuracy of the predictions computed by SVD. That is why it is common in the literature to impute the sparse R before computing its SVD. There are several imputation techniques and here are the most common one (Ghazanfar & Bennett, 2012):

1. **Impute by Zero (ByZero):** which fills all the missing entries by zero. However, this leads to predicting ratings close to zeros because of the abundance of zeros in the imputed matrix.
2. **Impute by Item Average (ItemAvgRating):** which fills the missing values in each column by the average rating of the corresponding item.
3. **Impute by User Average (UserAvgRating):** which fills the missing values in each row by the average rating of the corresponding user.

4. **Impute by the mean of ItemAvgRating and UserAvgRating (Mean_ ItemAvgRating&UserAvgRating):** which fills each missing value by the average of its corresponding item's average rating and its corresponding user's average rating.

As a result, a dense matrix R_{filled} is obtained and this could be interpreted as overcoming the sparsity problem associated with RSs. Furthermore, R_{filled} is normalized by subtracting the average rating of each user from its corresponding row resulting in R_{norm}. The last step is useful in offsetting the difference in rating scale between the different users (Vozalis & Margaritis, 2006).

At this point, SVD could be applied to R_{norm} to compute U_k (this holds users' features), S_k (holds the strength of the hidden features) and V_k (holds items' features) such that their inner product will give the closest rank-k approximation to R_{norm}. This lower-rank approximation of user-item matrix is better than the original one since SVD eliminate the noise in the user-item relationship by discarding the small singular values from S (Sarwar *et. al.*, 2000).

Henceforth, the preference of user i to item j could be predicted by the dot product of their corresponding features vectors; *i.e.*, compute the dot product of the ith row of $(U_k.S_k)$ and jth column of $V^T{}_k$ and add back the user average rating that was subtracted while normalizing R_{filled}. This could be expressed as:

$$P_{ij} = \bar{r}_i + \left(U_k.S_k\right)_{i,\_}.V^T_{\_,j}$$

Where p_{ij} is the predicted rating for user i and item j, $\bar{r}_i$ is the user average rating, $V^T_{\_,j}$ is the j^{th} column of V^T and $(U_k.S_k)$ is the i^{th} row of the matrix resulting from multiplying U_k and S_k.

In point of fact, the dot product of two vectors measures the cosine similarity between them. Thus, the above formula could be interpreted as finding the similarity between user i and item j vectors and then adding the user average rating to predict the missing rating $p_{ij.}$

EXPERIMENTS AND EVALUATIONS

Experimental Environment

All the experiments, behind this chapter, were implemented using Scala programming language on Eclipse, running on MacBook Pro with X 10.9.3 OS, 2.4 GHz Intel Core i5 processor and 8 GB of RAM. This machine served as a single node cluster for Apache hadoop 2.4.0 which was configured in pseudo-distributed mode. Moreover, Apache spark v. 1.0.2 was used as it provides fast distributed computations.

Data Set

The data set used in this work is the 1M MovieLens set which contains 1million ratings provided by more than 6000 users to around 3900 movies in the form of tuple (userID, MovieID, rating, timestamp). Ratings take integer values in the interval [1, 5] indicating how much the user likes the movie.

The aforementioned data set was divided into training set and test set based on different ratios (i.e. training ratios). Furthermore, the training set was used to fill the user-item matrix R of size 6040×3900;

which will be used to compute SVD, come-up with *U*, *S* and *V* matrices as well as predict ratings for unrated items. On the other hand, the test set was used to evaluate the accuracy of the predicted ratings.

Evaluation Metric

Different empirical evaluation metrics are there to assess the quality of the estimated predictions. The most common metrics are the statistical one such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). In this chapter, MAE is used.

The evaluation process of this work started by dividing the data set into two disjoint sets; one for training and the second for testing the system. The predicted ratings will be compared with the actual ratings in the test set by measuring MAE which will compute the average of the absolute difference between each predicted value and its corresponding actual rating (Sarwar *et al.*, 2000), *i.e.*

$$MAE = \frac{\sum_{I=1}^{N} \left| p_{i,j} - r_{i,j} \right|}{N}$$

Where N is the size of the test set, $p_{i,j}$ is the predicted rating for user i and $r_{i,j}$ is the actual ratings for user u. A smaller value of MAE refers to a higher prediction accuracy and thus better recommendations.

Choosing the Number of Dimensions

Reducing the dimensions of the original matrix R is useful because it aids in eliminating the noise and focusing on the important information. With this in mind, an appropriate value of k should be selected such that it can filter out the noise but not leads to the loose of important information. In another word, the value of k should be large enough to ensure capturing the essential structure of matrix R but small enough to filter out noise and avoid overfitting (Sarwar *et al.*, 2000; Berry, Dumais, & O'Brien, 1995). The best value of k will be experimentally determined by trying different values.

Experiments and Results

Our work started by loading 1M MovieLens data set into HDFS, where part of it, *i.e.* the training set, was used to fill the user-item matrix R. After that, R underwent two preprocessing operations: imputation and normalization. The imputation was done by Mean_ItemAvgRating&UserAvgRating, i.e. mean of item average rating and user average rating, after experimentally proving its superiority over other imputation techniques (refer to Experiment 2). Furthermore, the normalization step subtracted the average rating of each user from its corresponding row resulting in R_{norm}.

This was followed by using Apache Spark to compute SVD and come up with: U_k, V_k, and S_k. This is equivalent to extracting both user's and items' features from R. For that purpose, k was set to 20 after the results of experiment 1.

In order to compute a missing rating for one user, its corresponding row of ($U.S$) was multiplied by V^T column that corresponds to the target item and then denormalized by adding the user average rating.

Experiment 1: Determining the Appropriate Number of Dimensions

The work was executed several times with different values of k, ranging from 10 to 300, and different training ratio denoted as x. The results are presented in table 1 by demonstrating the value of MAE obtained for each pair of (k, x) after imputing the rating matrix by the mean of user average rating and item average rating .

As a result, 20 is found to be a good value of k and thus it is fixed in the other experiments.

Experiment 2: Determining the Best Imputation Technique

Four techniques of imputation (i.e. impute ByZero, impute by ItemAvgRating, impute byUserAvgRating and impute by Mean_ItemAvgRating&UserAvgRating) were tested against multiple values of k. The results presented in table 2 shows the superiority of imputing by Mean_ItemAvgRating&UserAvgRating as it gives the lower values of MAE. Thus, it will be used in the other experiment.

Experiment 3: Evaluating the Accuracy of SVD on different training ratio

The work was executed several times with different training ratios ranging from 35% to 95%, with k =20 and k = 100. Note that the user-item matrix R was imputed by Mean_ItemAvgRating&UserAvgRating before computing its SVD. This experiment intended to study the real effect of the number of dimensions k on the prediction quality. Its results are presented in table 3.

Table 1. MAE Measured for different training ratios and different values of k

	Training Ratio		
k	**x= 40%**	**x= 60%**	**x= 80%**
10	0.752063387	0.740626631	0.729999025
20	0.751235952	0.737954414	0.724461829
30	0.751605786	0.738107001	0.723582636
40	0.752364535	0.738714968	0.723803209
50	0.753327523	0.739710017	0.725192112
60	0.754025958	0.740751258	0.726154802
70	0.754724263	0.741471676	0.727282644
80	0.755378978	0.742729653	0.728420107
90	0.755930363	0.743485881	0.729623117
100	0.756510921	0.744293948	0.730338578
150	0.759248743	0.748479721	0.735940524
200	0.761302063	0.752022533	0.740869085
250	0.762834111	0.754352648	0.744583552
300	0.764142161	0.756565002	0.747871699

Table 2. MAE for Different Imputation Techniques and Different Values of k

Imputation Technique				
k	**ByZero**	**UserAvgRating**	**ItemAvgRating**	**Mean_ UserAvgRating&ItemAvgRating**
10	1.287082589	0.763900832	0.738045498	0.729999025
20	1.340593039	0.758800163	0.732242767	0.724461829
30	1.351316283	0.758681267	0.731619885	0.723582636
40	1.342134276	0.760268678	0.731981172	0.723803209
50	1.331709429	0.762213489	0.733055102	0.725192112
60	1.317791963	0.764543289	0.733818382	0.726154802
70	1.303016117	0.766207466	0.734838138	0.727282644
80	1.288611141	0.768863197	0.736044939	0.728420107
90	1.274236198	0.770501176	0.737138793	0.735940524
100	1.261225796	0.772165966	0.738514469	0.730338578
150	1.198043117	0.780474068	0.744252067	0.735940524

Table 3. MAE for different training ratios with k=20 and k=100

	k	
Training Ratio x	**k=20**	**k=100**
0.35	0.7549804	0.759679233
0.4	0.751235952	0.756510921
0.45	0.747814989	0.753365509
0.5	0.744018997	0.749880733
0.55	0.741011898	0.747432557
0.6	0.737954414	0.744293948
0.65	0.734098595	0.740644033
0.7	0.731331182	0.737610654
0.75	0.727741033	0.734113758
0.8	0.724461829	0.730338578
0.85	0.721460479	0.727461993
0.9	0.72096045	0.726117655
0.95	0.733633592	0.740065812

The figure compares the MAE values of different training ratio for two values of *k*. It is obvious that MAE takes less values for *k=20* than those of *k = 100*; but not significant. This is validating what has been stated repeatedly in the literature; that a larger value of *k* does not necessarily gives better predictions despite the fact that it gives a closer approximation to the original matrix.

DISCUSSION

The main computational advantage of running these experiments, which implement SVD-based recommender system, using Hadoop, Spark and Scala is its easy parallelization. This proves the powerfulness of these frameworks/APIs in implementing large-scale systems with parallelized operations in distributed mode.

The results are comparable with the results of other works, discussed previously in the literature review, such as the one conducted by Sarwar and his colleagues, by Gong and Dai as well as by Zhou and his colleagues; that were carried on significantly smaller data set (i.e. 100 K MovieLense Data Set). This proves that SVD approach is not only effective for, ordinary, small data but even for Big Data sets.

Indeed, this work resulted in better predictions when compared with Sarwar *et. al.* work although it has been carried on much bigger data set. To put it differently, the best predictions obtained by Sarwar *et. al.* on 100K MovieLense data set were using training data set of 80% and $k \in [20,100]$ as they get MAE ranging from 0.748 to 0.732 (Sarwar, *et al*, 2000). However, the MAE obtained by our implementation, for the same values of k, the same training ratio and 1M MovieLense data set, were ranging between 0.7230 and 0.723.

While looking for the best value of k, 20 was found as the favorable one since it gave a small value of MAE when checking it over different training ratios. This is reasonable when comparing it with previous works which found $k = 14$ (Sarwar, *et al*, 2000; Sarwar *et. al.*, 2002) or $k = 15$ (Gong, Ye & Dai, 2009) for smaller data set. Notable, increasing the volume of the data set to 1 million ratings did not, dramatically, increase the value of k which validates other researchers' opinions, reported in some research papers, in which a small number of dimensions usually give pretty good results with good approximation to the original matrix R. This is simply because a small value of k is sufficient to capture the important features of users and items and thus make good predictions. However, increasing the value of k might simply represent adding more noise to the data which does not add value to the process of making predictions.

Furthermore, trying different imputation techniques and tracking their MAE showed the importance of pre-processing steps and its effect on the prediction accuracy. As per our experiments, Mean_ItemAvgRating&UserAvgRating outperformed other imputation techniques since it gave lower MAE.

Moreover, repeating the experiments multiple times with different values of k and different values of training ratio x; revealed the sensitivity of the prediction quality to the sparsity of the data set since MAE values decrease as the training ratios increase and the sparsity decrease. Added to that, it revealed the significant effect of the value of k on the prediction quality, as well as the effectiveness of SVD in dealing with cold-start cases.

FUTURE RESEARCH DIRECTIONS

The work presented in this chapter is just the starting point in exploring the current state of recommender system in Big Data Era. One might extend the work by deploying the system on multi-node cluster in order to be able to assess its scalability, performance and accuracy in a distributed mode. Another future direction could be to implement a hybrid approach of SVD that combines stochastic version of SVD proposed by Lee and Chang (Lee & Chang, 2013), incremental version of SVD proposed by Sarwar, B. et al. (Sarwar, B. et al., 2002) and Expectation Maximization technique presented by Kurucz et al. (Kurucz

et al., 2007). This will be an iterative process that applies a stochastic version of SVD, repeatedly, to a matrix and use the outcome of one iteration to impute the input of the next iteration. Stochastic SVD could be done in an incremental manner such that the advent of a new user will not imply re-computing the decomposition of user-item matrix; but the new user will be projected to the existing SVD model.

CONCLUSION

As recommender systems proved their powerfulness in personalizing the web content for each user, more research efforts have been devoted to improving it and evaluate its different algorithms. The commonly adapted approach is collaborative filtering that mines the interaction records between users and items to infer user's taste and thus recommends items that match his taste. Surprisingly, CF techniques have started facing some challenges with the dawn of Big Data era. This new phenomenon is inflaming the data volume to be processed by RS and thus raises some concerns about the sparseness of the available data, scalability of RSs as well as the quality of the predictions. As a consequence, new approaches of CF have been proposed and studied such as Singular value decomposition. In addition, various Big Data frameworks and APIs (such as Hadoop, Mahout and Spark) have been released and tried in building large-scale recommender systems.

This chapter contributes to the state of the art of recommender systems as it provides an implementation of a large scale SVD-based recommender system using both Apache Hadoop and Spark. This came as a result of an intensive study to the literature as well as conducting several experiments using Scala programming language on top of apache Hadoop and Spark. The study involved several topics which are: Big Data phenomenon, the different techniques and approaches of recommender systems together with their pros and cons, the challenges posed by big data on recommender systems and CF in particular, the applicability of SVD for recommender systems as well as its effectiveness in solving the aforementioned challenges. The experiments were conducted to determine the optimal values of two essential parameters that affect SVD-based RS which are: the imputation technique to be used in filling the user-item matrix before processing it and the number of dimensions to be retained after decomposing the matrix. The results showed that the best imputation technique is using the average of both item average rating and user average rating and that the optimal number of dimensions is k=20 as it gave the lowest MAE.

This work solved the scalability problem by utilizing Hadoop and its valuable features. In addition, it showed that pretty good quality could be achieved by choosing a robust imputation technique (as a preprocessing step) before applying SVD to the user-item matrix. Moreover, it asserted that Apache Spark comes with attractive merits which enable easy integration with Hadoop and easy development of parallelizable code.

This drew a conclusion that a careful implementation of a scalable SVD-based collaborative filtering recommender system is effective when choosing the right parameters and the appropriate frameworks and APIs.

REFERENCES

Berkeley School. (2012). *Lecture 12 -Analyzing Big Data with Twitter: Recommender Systems*. Retrieved February 10, 2014, from http://www.youtube.com/watch?v=NSscbT7JwxY

Berry, M. W., Dumais, S. T., & O'Brien, G. W. (1995). Using linear algebra for intelligent information retrieval. *SIAM Review*, *37*(4), 573–595. doi:10.1137/1037127

Bizer, C., Boncz, P., Brodie, M. L., & Erling, O. (2011). The Meaningful Use of Big Data: Four Perspectives - Four Challenges. *SIGMOD*, *4*(4), 56 – 60. Retrieved from http://www.sigmod.org/publications/sigmod-record/1112/pdfs/10.report.bizer.pdf

Baker, K. (2013). *Singular Value Decomposition Tutorial*. Retrieved December 26, 2014, from http://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf

Bloor Research. (2012). *Big Data Analytics with Hadoop and Sybase IQ*. London: Bloor.

Buros, W. M. (2013). *Understanding Systems and Architecture for Big Data*. IBM Research Report. Retrieved December 23, 2014 from http://cnf-labs.org/pdf/research-report-ibm.html#

Casey, E. (2014). *Scalable Collaborative Filtering Recommendation Algorithms on Apache Spark*. (Unpublished Senior Thesis). Claremont Mckenna College, USA.

Chen, J., Chen, Y., Du, X., Li, C., Lu, J., Zhao, S., & Zhou, X. (2013). Big Data Challenge: A data management perspective. *Frontiers of Computer Science*, *7*(2), 157-164. Retrieved from http://iir.ruc.edu.cn/~jchchen/FCSBigData.pdf

Chiky, R., Ghisloti, R., & Aoul, Z. K. (2012). Development of a distributed recommender system using the Hadoop Framework. In *Proceedings of EGC 2012* (pp. 495 – 500).

Cloudera & Teradata. (2011). *Hadoop and the Data Warehouse: When to Use Which*. Cloudera, Inc. and Teradata Corporation.

Databricks. (2014). *Intro to Apache Spark*. Retrieved October 24, 2014, from http://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf

Dittrich, J., & Ruiz, J. A. Q. (2012). Efficient Big Data Processing in Hadoop MapReduce. In *Proceedings of VLDB Endowment* (pp. 2014-2015). VLDB Endowment. doi:10.14778/2367502.2367562

Fayyad, U. M. (2012). Big Data Everywhere, and No SQL in sight. *SIGKDD Explorations*, *14*(2), 1–2. Retrieved from http://www.sigkdd.org/sites/default/files/issues/14-2-2012-12/V14-02-01-editorial.pdf

Ghazanfar, M. A., & Bennett, A. P. (2012). The advantage of Careful imputation source in sparse data-environment of recommender systems: generating improved SVD-based recommendations. *Informatica, 37*, 61-92. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.368.2112&rep=rep1&type=pdf

Gong, S., Ye, H., & Dai, Y. (2009.) Combining Singular Value Decomposition and Item-based Recommender in Collaborative Filtering.*Second International Workshop on Knowledge discovery and data mining*, (pp. 769-722). doi:10.1109/WKDD.2009.132

Gower, S. (2014). *Netflix prize and SVD*. Retrieved December 20, 2014, from http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf

Hortonworks. (2012). *Apache Hadoop: The Big Data Refinery.* Hortonworks.

Kabore, S. C. (2012). *Design and implementation of a recommender system as a module for Liferay portal*. (Unpublished master dissertation). University Polytechnic of Catalunya (UPC), Spain.

Kluver, D., & Konstan, J. A. (2014). Evaluating Recommender Behavior for New Users. *RecSys*, *14*, 121–128.

Lee, C. R., & Chang, Y. F. (2013). Enhancing Accuracy and Performance of Collaborative Filtering Algorithm by Stochastic SDV and Its MapReduce Implementation. *2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum*, (pp. 1869-1878).

Melville, M., & Sindhwani, V. (2010). Recommender Systems. In *Encyclopedia of Machine Learning* (pp. 829-838). http://vikas.sindhwani.org/recommender.pdf

Odersky, M., Spoon, L., & Venner, B. (2008). *A Scalable Language*. Retrieved December 15, 2014, from http://www.artima.com/scalazine/articles/scalable-language.html

Owen, S. (2012). *Mahout in Action*. New York: Manning Publications.

Pagare, R., & Patil, S. A. (2013). Study of Collaborative Filtering Recommendation Algorithm – Scalability Issue. *International Journal of Computers and Applications*, *67*(25), 10–15. doi:10.5120/11742-7305

Papagelis, M., Rousidis, L., Plexousakis, D., & Theoharppoulous, E. (2005). Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms. In M. Said Hacid, N. V. Murray, Z. W. Ras, & S. Tsumoto (Eds.), *Foundations of Intelligent System*. New York: Springer Berlin Heidelberg.

ParAccel Inc. (2012). *Hadoop's Limitations for Big Data Analytics*. ParAccel; doi:10.1007/11425274_57

Quora. (2011). *LinkedIn Recommendations: How does LinkedIn's recommendation system work?* Retrieved April 28, 2014, from http://www.quora.com/LinkedIn-Recommendations/How-does-LinkedIns-recommendation-system-work

Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to Recommender Systems Handbook. In F. Ricci et al. (Eds.), *Recommender Systems Handbook*. New York: Springer. doi:10.1007/978-0-387-85820-3_1

Sarwar, B. (2000). Application of Dimensionality Reduction in Recommender System -- A Case Study. *ACM WebKDD 2000 (Web-mining for Ecommerce Workshop).* Retrieved from http://files.grouplens.org/papers/webKDD00.pdf

Sarwar, B. (2001). Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (pp. 285-295). New York: ACM.

Sarwar, B. (2002). Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems. In *Proceedings of Fifth International Conference on Computer and Information Science* (pp. 27- 28).

Schelter, S., & Owen, S. (2012). Collaborative Filtering with Apache Mahout. RecSysChallenge. Dublin: ACM. Retrieved from http://ssc.io/pdf/cf-mahout.pdf

Schönberger, V. M., & Cukier, K. (2013). *Big Data: A revolution that will transform how we live, work, and think*. New York: Houghton Mifflin Harcourt.

Soft Layer. (2014). *Big Data Hosting*. Retrieved December 30, 2013, from: http://www.softlayer.com/solutions/big-data/

Spark Summit. (2014). *Spark Summit 2014*. Retrieved October 16, 2014, from: http://spark-summit.org/2014

Spark 1.2.0. (2014). *Spark Programming Guide.* Retrieved October 24, 2014, from: http://spark.apache.org/docs/latest/programming-guide.html

Thangavel, S. K., Thampi, N. S. & I, J. C. (2013). Performance Analysis of Various Recommendation Algorithm Using Apache Hadoop and Mahout. *International Journal of Scientific and Engineering Research, 4*(12), 279-287. Retrieved from http://www.ijser.org/researchpaper%5CPerformance-Analysis-of-Various-Recommendation-Algorithms.pdf

TutorialsPoint. (2014). *Scala Tutorial*. Retrieved December 22, 2014, from: http://www.tutorialspoint.com/scala/scala_pdf_version.htm

Ullman, J. D. (2012). *Designing Good MapReduce Algorithms.* Retrieved May 5, 2014, from: http://xrds.acm.org/article.cfm?aid=2331053

Villa, A. (2012). *Transfering Big Data Across the Globe*. University of New Hampshire Durham.

Vozalis, M. G., & Margaritis, K. G. (2006). Applying SVD on generalized Item-based filtering. *International Journal of Computer Science & Application, 3*(3), 27-51. Retrieved from http://www.tmrfindia.org/ijcsa/v3i34.pdf

Walunj, S. & Sadafale, K. (2013a). Price based Recommendation System. *International Journal of Research in Computer Engineering and Information Technology, 1*(1).

Walunj, S., & Sadafale, K. (2013b). An Online Recommendation System for E-commerce Based on Apache Mahout Framework. In *Proceedings of the 2013 annual conference on Computers and people research* (pp. 153-158). New York: ACM. doi:10.1145/2487294.2487328

Ward, J. S., & Barker, A. (2013). *Unified By Data: A Survey of Big Data Definitions*. Retrieved December 15, 2014, from: http://arxiv.org/pdf/1309.5821v1.pdf

White, T. (2011). *Hadoop: The definitive Guide*. O'Reilly Media.

Zhao, Z. D., & Shang, M. S. (2010). User-based Collaborative-Filtering Recommendation Algorithms on Hadoop. In *Proceedings of Third International Conference on Knowledge Discovery and Data Mining* (pp. 478-481). IEEE.

Zikopoulous, P. C. (2012). *Understanding Big Data: Analytics for Enterprise Class; Hadoop and Streaming Data*. New York: McGraw Hill.

Zhou, X., He, J., Huang, G., & Zhang, Y. (2012). A personalized recommendation algorithm based on approximating the singular value decomposition (ApproSVD). *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. IEEE.

KEY TERMS AND DEFINITIONS

Apache Hadoop: An open source framework that provides a distributed data storage and computation utilizing commodity servers.

Apache Spark: An open source framework supporting in-memory data analysis and distributed programming that has been built on top of Apache Hadoop to facilitate distributed computation.

Column-Orthonormal Matrix: A matrix where all its vectors length is 1 and the dot of any two of them is zero since they are orthogonal.

Matrix-Rank: The number of rows that are linearly independent.

Matrix-Transpose: A matrix resulting from swapping row-vectors with column vectors.

Recommender System: An automated system that suggests relevant, not seen yet, items to the user.

Scala: A programming language that facilitate the development of scalable systems.

Singular Value Decomposition: Breaking down a matrix into three, smaller matrices, U, S and V in which their product yields back the original matrix.