Movie Recommendation System by Distributed Stochastic Gradient Descent

Yuchen Shen

Guancheng Yao

Abstract

Recommendation System is utilized in many areas including music, books and etc. It is very important for business to recommend potential charming items to users, especially when the number of items are too much enormous. Nowadays, large scale data always stores in distributed file system like HDFS, and traditional stochastic gradient descent based matrix factorization will not perform well for large scale matrix and will not be applicable for parallelized systems. In this study, we present a method, distributed SGD (DSGD) to solve matrix factorization based recommendation system for movies. First, we introduce "stratified" SGD variant (SSGD) that applies to general loss-minimization problems in which the loss function can be expressed as a weighted sum of "stratum losses". Then specialize SSGD to obtain a new matrix-factorization algorithm, called DSGD, that can be fully distributed and run on large-scale datasets using Spark. Matrix factorization is speed up by using DSGD and stochastic gradient descent with many iterations will benefit from using Spark. Experiments suggest that DSGD in Spark converges significantly faster and has better scalability properties than prior algorithms.

Key Word: Recommendation System, Matrix Factorization, Distributed Stochastic Gradient Descent

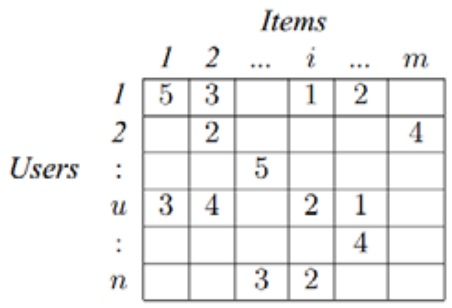
1. Introduction & Background

With the development of online business, people are suffering from information overload when go over the items. Online businesses have enormous number of items providing to users, it is difficult for users to find preferred information or items; and also, it is hard for businesses to attract user’s attention to some items. Recommendation system is a core component for these online business, such as Netflix, Amazon and Apartment.com. The goal is to generate relation between users and items, helping user to find charming items and helping business show items to uses who is probably interested in. Today, many businesses use big data to make super relevant recommendations and growth revenue.

2. Overview of recommendation system

2.1 What does a recommendation system do in principle?

When we know some users' evaluation of certain products, we can use these existing reviews to determine the user interests in other products. General speaking, a recommendation system can predict the missing values of the “User-Item” scoring matrix.



		Items					
		1	*2*	...	*i*	...	*m*
	1	5	3		1	2	
	2		2				4
Users	:			5			
	u	3	4		2	1	
	:					4	
	n			3	2		

Figure 1. User-Item scoring matrix.

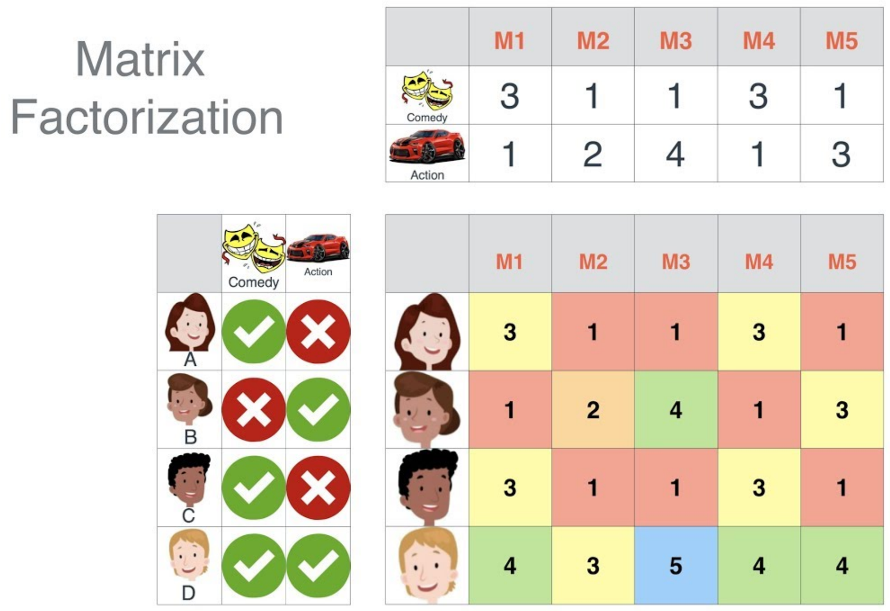
2.2 Method of filling up missing values:

i. User-based method: Based on people like you, recommend items that others like.

ii. Item-based method: Recommend to users something similar to what they liked before.

iii. Matrix factorization based

- Each user can be described by k attributes or features. For example, feature 1 might be a number that says how much each user likes science fiction movies.
- Each item (movie) can be described by an analogous set of k attributes or features. To correspond to the above example, feature 1 for the movie might be a number that says how close the movie is to pure science fiction.
- If we multiply each feature of the user by the corresponding feature of the movie and add everything together, this will be a good approximation for the rating the user would give that movie.



	M1	M2	M3	M4	M5
Comedy	3	1	1	3	1
Action	1	2	4	1	3

	Comedy	Action
A	✓	✗
B	✗	✓
C	✓	✗
D	✓	✓

	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

Figure 2. Matrix factorization

Recommendation system always based on several different algorithms. With implementing the method above, Netflix could notify users the movie they might like, which will increase customer loyalty and increase the chance that customers continue subscribing their membership. In Amazon, the recommendation system plays a vital role in increasing sales. In fact, the product market follows the long tail effect. From the perspective of people's needs, most of the demand will be concentrated in the head, and this part can be called popular, while the demand distributed in the tail is personalized, scattered and small demand. Amazon needs to recommend users with those products which are not popular, or nobody will find that product and buy it. To sum up, a good recommendation system determines whether the company will have considerable income.

In this paper, we will introduce the matrix factorization based algorithm. With the increasing of data scale, distributed file system is now popular used, and data mining method is now modifying to fit in distributed algorithm including matrix factorization based recommendation system.

3. Naïve Solution

So far, we have briefly introduced several methods to implement a recommendation system. The simplest way is user-based and item-based method.

3.1 The user-based method

Calculating the similarity between users by cosine similarity. Suppose we have user *u* and *v*, *N(u)* denotes the set of items that user u likes:

$$a_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)||N(v)|}}$$

According to this equation we can get the matrix of user similarity. And then, it will use the weighted value of the evaluation of the product *i* from the user among the top *k* similar users who have evaluated the product *x* as the recommendation to the user for the product *i*. Suppose we want to know user u's interest on item *x, p*(*u, x*) should be his predicted rating on item *x*:

$$p(u,x) = \sum_{v \in S(u,k) \cap M(x)} a_{uv} r_{vx}$$

Here, the *S*(*u, k*) denotes the top *k* most similar users to user *u*. *M*(*x*) is the union of users who like item *x*, *αuv* is the similarity value between users. And the *γvx* is the rating that user *v* gave to item *x*.

But user based method has its draw backs. It is hard to interpret the result to customers based on the result we have. Also with the increasing of registered users, it is more and more difficult to calculate the similarity matrix between users.

3.2 The item-based method

This method is very similar to the user-based method above. Where user-based is row-oriented, item-based method is a column-oriented approach. Suppose we have item *i* and *j*, *M*(*i*) denotes the set of users who like item *i*, then we can get similarity between items:

$$b_{ij} = \frac{|M(i) \cap M(j)|}{\sqrt{|M(i)||M(j)|}}$$

Here, through analyzing the behavior of users, the item-based method records similarities between items. So, the similarity between items come from the behavior of users. In the second step, the recommended score of item *x* is calculated by the weighted value, which were given by the other users, of the top *k* items which are most similar to item *x*, and this score is the degree of recommendation to the user for the item *x*. And below the equation of predicting the score user *u* will give to item *x*.

$$p(u,x) = \sum_{i \in Q(x,k) \cap N(u)} b_{xi} r_{ui}$$

To sum up, the user-based collaborative filtering is more socialized. It focuses on reflecting the interests of a small number of group. And the user will get recommended information according to the interests of other group members in that group. In this situation, the user-based CF is more suitable for the situation when the item update speed is much faster than the user update speed. Because it only maintains the user similarity matrix. Where item-based collaborative filtering is much more personalized, it focused more on the history record of a single user. Item-based CF focuses on maintaining the historical interest of users, it reflects the user's own interests. As a result, item-based CF is suitable for the situations where item update is not fast, which is the actual Internet situation. In current internet environment, the user base is huge and it increases rapidly. Compared to the number of users, the number of products is relatively small. So, the item-based CF is used in a lot of shopping websites.

3.3 Alternating Least Square (ALS)

ALS achieves matrix factorization by gradient decent method. As we introduced above, we can get two matrices after matrix factorization. ALS rotates between fixing one of the unknown user latent matrix or item latent matrix. When one is fixed the other can be computed by solving the least-squares problem. This approach is useful because it turns the previous non-convex problem into a quadratic that can be solved optimally [2]. The basic idea is Stochastic Gradient Descent, which we will introduce later.

4. Proposed Solution

As introduced above, previous solutions have many constrains and disadvantages, ALS requires double-partitioning and each of the factor matrices must fit in main memory. This will definitely limit the data scales that can be handled and meanwhile require large memory infrastructures. In this section, we will introduce a stratified stochastic gradient descent (SSGD) that will help to construct efficient distributed SGD. It can distribute both the fact matrix and factor matrix to parallel computation, and no data transformation between workers where generates huge cost.

4.1 Stochastic Gradient Descent based Matrix Factorization

In small-scale SGD based Matrix Factorization, for each iteration, we select a single data point (*i, j*) and update the corresponding row *i* of factor matrix *W* and column *j* of factor matrix *H* in the direction of negative gradient.

$$\mathbf{W}'_{i*} \leftarrow \mathbf{W}_{i*} - \epsilon_n \frac{\partial}{\partial \mathbf{W}_{i*}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$
$$\mathbf{H}'_{*j} \leftarrow \mathbf{H}_{*j} - \epsilon_n \frac{\partial}{\partial \mathbf{H}_{*j}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$$

Repeat the process, until match the termination metric. However, in larger scale matrix, this algorithm is not work well again, since the huge number of data point will deteriorate the updating process in each iteration.

Simple distributed the matrix data point into parallel computation is not acceptable. Considering two data point x_{11} andx_{12}; when parallel these two points, we will update W_{1*}, H_{*1} using x_{11}, and at the same time update W_{1*}, H_{*2} using x_{12}. Then W_{1*} will be updated simultaneously, and will affect the result of each other. Therefore, we will present an algorithm that can distributed data point (partitions) to update factor matrix in parallel way.

4.2 Stratified Stochastic Gradient Descent

In matrix factorization, SGD is to minimize the sum of local losses L to make factor matrix W*H as close to original matrix V as possible. In larger scale distributed system, the solution is dividing the matrix into stratum for each iteration and performing sequential stochastic gradient descent within each stratum in parallel.

In distributed calculation, losses need to be measure separately. Stratified stochastic gradient descent is the solution. Losses L is divided into sum of weighted loss as:

$$L = w_1 L_1 + w_2 L_2 + \cdots + w_q L_q \,, \qquad 0 < w_i < 1, \sum\nolimits_i w_i = 1$$

In practice, each stratum often corresponds to a partition of the distributed large scale data, and total loss L is the sum of weighted loss in each partition. SSGD runs standard stochastic gradient descent on a single stratum at a time, but switches stratum in a way that guarantees correctness. To confirm convergence of SSGD, some appropriate regularity conditions need to be satisfied, but already satisfied in most matrix factorization problem which guarantee the generality of SSGD.

4.3 Distributed Stochastic Gradient Descent

Distributed stochastic gradient descent is a specialized version of SSGD algorithm. Choosing appropriate stratum such that SGD can be run on each stratum in a distributed manner. Generally speaking, distributed SGD is not easy to implement, since θ_{n+1} is depend on θ_n, $\theta_{n+1} = \theta_n - \varepsilon L'$. However, SSGD has some structure properties can be used in distributed matrix factorization.

i. Interchangeability

The most important property of SSGD is interchangeability, which guarantees that well split matrix blocks can be performed parallel in Spark workers without communication cost.

Given a pair of element in matrix V, (i,j) and (i',j') is interchangeable, if $i \neq i'$ and $j \neq j'$. As introduced above, local loss L_{ij} is only dependent on the truth value V_{ij}, the i^{th} row and j^{th} column of factor matrix W and H, so $L_{i'j'}$ will not depend on any element in W_{i*} or H_{*j}. In other word, $l\ (V_{ij}, W_{i*}, H_{*j})$ and $l\ (V_{i'j'}, W_{i'*}, H_{*j'})$ which the SGD is updating, do not depend on any overlapping elements in the matrix and can be performed in parallel.

Similar in more general cases, a set of elements in the matrix are interchangeable, if any pair of elements in the two sets are interchangeable. Therefore, when updating the local loss of these interchangeable elements, they can also be implemented parallel, because their losses are calculated by non-overlap partition of the fact matrix V and factor matrix W and H.

It is easy to extend the interchangeable property from between elements to between set of elements, for here in distributed file system, the set of elements in matrix can be thought as blocks of a matrix stored in HDFS. Consider I, I' and J, J' are sets of row and column indices of V. Matrix blocks IJ and $I'J'$ are interchangeable if $I \cap I' = \emptyset$ and $J \cap J' = \emptyset$, it follows that if two blocks of V share neither rows or columns, then the sets of training points contained in these blocks are interchangeable.

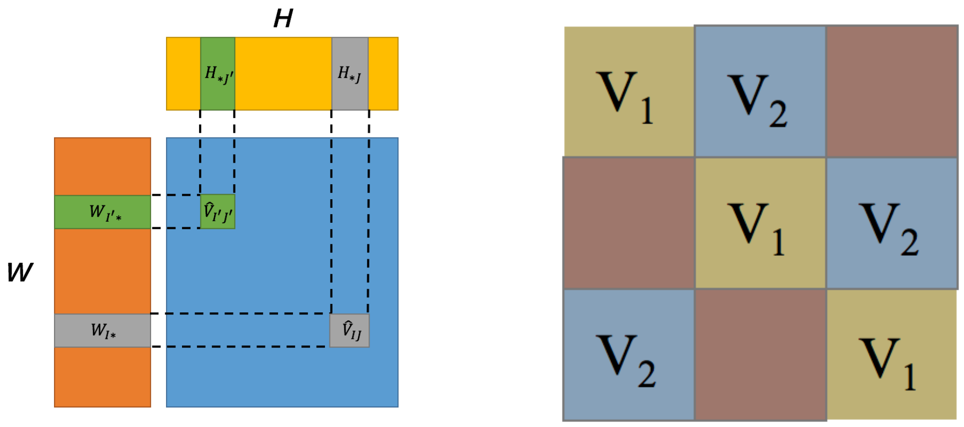


Figure 3. Interchangeability of blocks in a matrix.

ii. Distributed SGD Algorithm

As the reason above, we will not have to follow the sequence of SGD necessarily. Changing the order of interchangeable training points will not affect the final outcome.

Thus, if we choose a set of interchangeable blocks, then we can spill these training blocks into parallel computation to update parameters which only corresponding to that block, and collect the update to get the final result.

4.4 Implement in Spark

Since long iterative process is needed to minimize the loss, Spark will perform better than Hadoop map-reduce jobs. Spark will read HDFS files as RDD and stored in memory or disk, for each iteration, the partition of matrix will not to be read again and again. And also, we don't need to spill the matrix in advance, we can spill the matrix RDD just before the whole algorithm.

4.5 Workflow

The algorithm workflow shows below in *figure*, we use a simple case of *d* workers in Spark as example. In the current work, we perform *data-independent blocking*; First, read the matrix from HDFS as RDD, create $d \times d$ partitions of size $(m/d) \times (n/d)$ each; the factor matrices W and H are partitioned conformingly. For each iteration, if not converge, the algorithm chooses all different sets of interchangeable matrices blocks to perform SGD in parallel Spark workers. As a result, all partition will be used to update the parameter using SGD.

As shown in the figure, there are two workers. For the first iteration, they will receive Z_{11}, W_{1*}, H_{*1} and Z_{22}, W_{2*}, H_{*2}, Z_{12}, W_{1*}, H_{*2} and Z_{21}, W_{2*}, H_{*1}; Update the partition of parameters separately, we have used every partition $Z_{11}, Z_{12,} Z_{21}, Z_{22}$ to update W and H.

Finally, collecting all updated partitions of parameters, we will generate the final updated parameters.

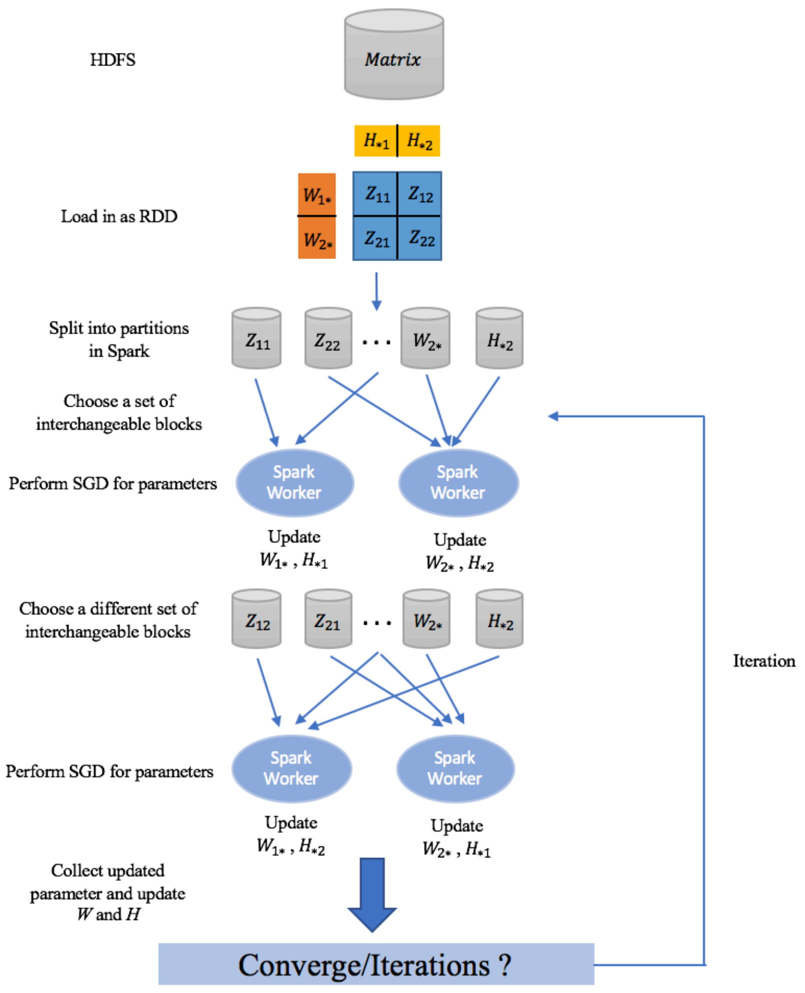


Figure 4. Example workflow of DSGD with 2×2 matrix blocks.

We also show an example process of DSGD that minimizing the loss in figure. The loss has been decomposed into two sets of interchangeable blocks. Each time, select one interchangeable partition to apply optimization. For many iterations, DSGD will converge to the final optimum.

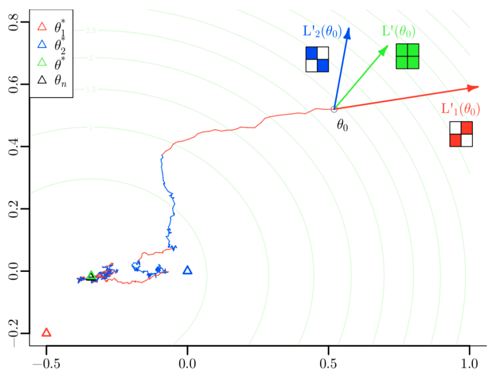


Figure 5. Example of DSGD with 2 workers.

5. Result

We plot the learning curve of the DSGD model in 100 iterations and this method converge in only a few iterations. The validation RMSE will keep static after 40 iterations when we set 12 latent factors and the penalty factor as 0.05 which was selected after cross validation.

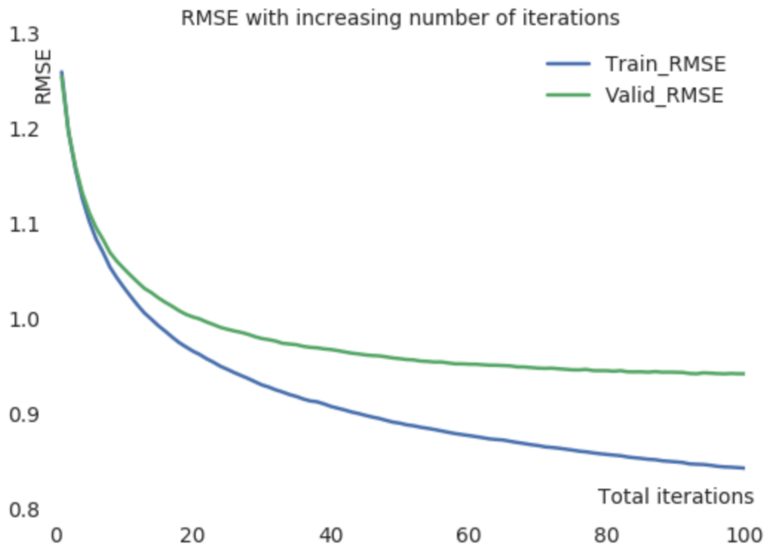


Figure 6. RMSE curve

Here we split it into 16 blocks and each 4 blocks can run parallel because of Interchangeability. In each job, there is only one stage because of Narrow Dependency and each stage has 4 workers running parallel. When we have 100 iterations, there are 400 jobs in total, which is got by (number of iterations (100) * number of stratums (4)). The speed of our DSGD algorithm is much faster than the traditional ALS.

The ALS runs too slow when there are 100 iterations and the running time is not compatible with DSGD's running time. Here is the result when there are 100 iterations.

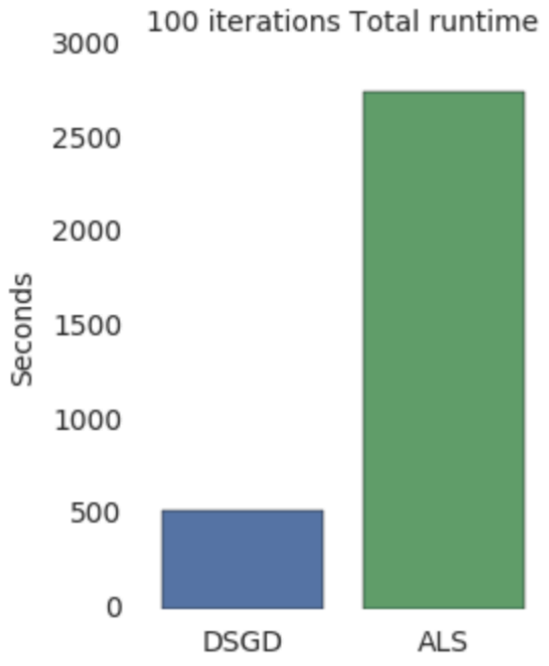


Figure 7. Running time comparison

In conclusion, the DSGD approach could achieve matrix factorization much faster without affecting accuracy.

Reference

[1]. https://www.youtube.com/watch?v=ZspR5PZemcs

[2]. Aberger, C.R., 2016. Recommender: An analysis of collaborative filtering techniques.

[3]. Gemulla, Rainer, et al. "Large-scale matrix factorization with distributed stochastic gradient descent." *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011.