README

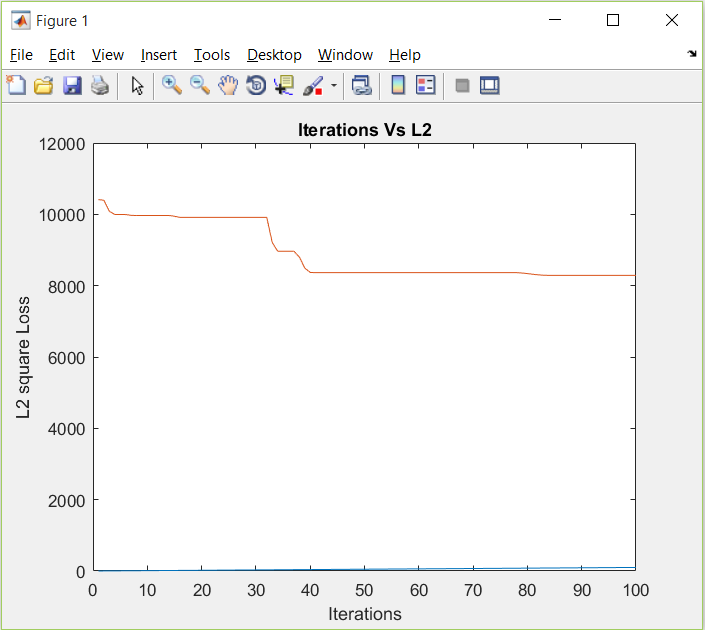
1. Place the src folder, input file nf\_subsample.csv and dsgd.sh script in the Spark

folder.

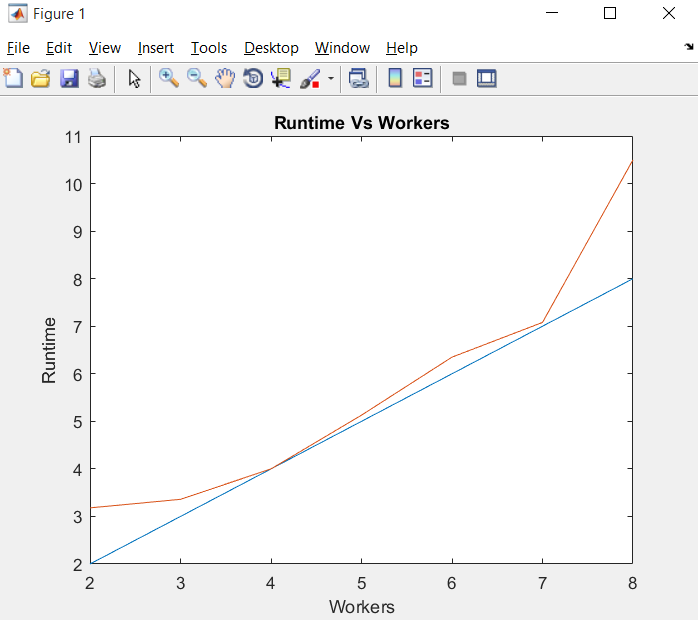
1. Run the dsgd.sh script from your Spark folder
2. Answers for Ques 1 to 7 are in this document. Their respective readings are in .txt files by names Question1.txt, Question2.txt and so on.
3. BONUS Question 1 and BONUS Question 2 are in .txt files by the same name. In it, first is the scala code and at the bottom is the result and answer to the question.

Q1) **L2 versus Iterations**. Please refer Ques1.txt for actual values.

The trend observed is a decreasing one. As iterations go ahead, the reconstruction error goes on decreasing. The more number of iterations, the more chance of getting a favorable value for epsilon due to which high probability of getting converged.

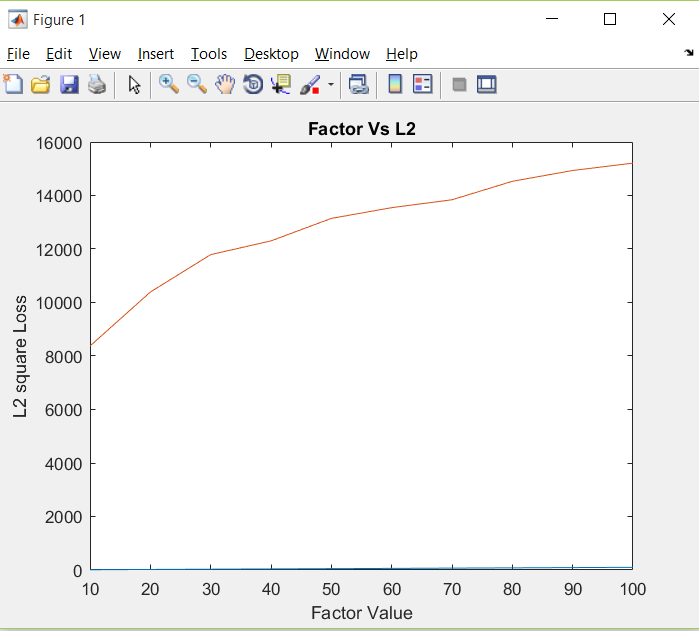


Q2) Runtime vs. Workers. Please refer Question2.txt for actual values. The trend observed is as number of workers increase, the runtime increases. (Please ignore the blue line).



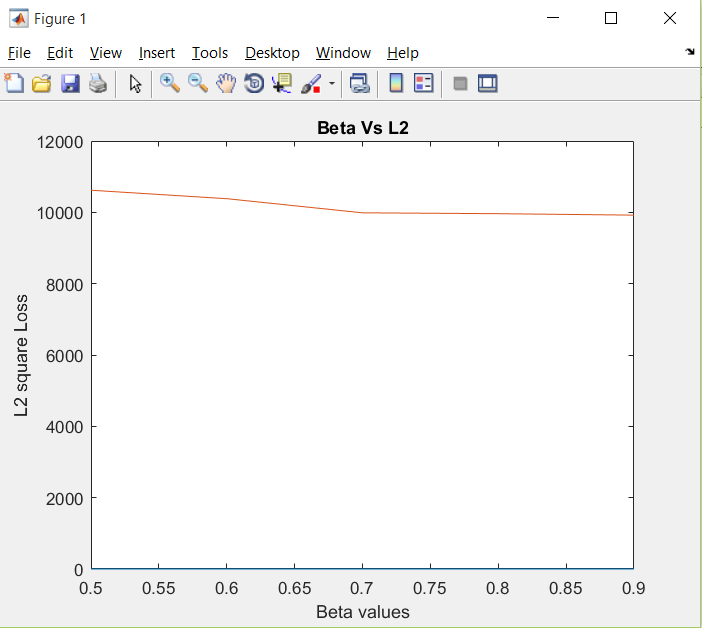
Q3)

Reconstruction error L2 versus Number of Factors (Rank) . Please refer Ques3.txt for actual values. The trend observed is as factors increase, the reconstruction error also increases.



Q4)

Graph of L2 versus Beta values. Please refer Question4.txt for actual values. The trend observed is, the reconstruction error decreases as Beta is increased. But the increase is very less.



Q5) In DSGD- MF we take initial values of W and H at random. DSGD differs in SVD at this point since we are not taking original values. Taking random initial values will help avoid reaching local minima. Customized step size after every iteration will help in convergence.

Moreover, the Sigma diagonal matrix values, as in in SVD, will be absorbed into W and H.

Q6) My method for DSGD

1) I converted given data of User,Movie,Rating to matrix form such that UserIDs are rows Movie IDs are columns and Ratings are values. I used the Rating function for that and created a Co-ordinate matrix. Then I converted it to breeze dense matrix because later on I have worked only with breeze matrices.

2) STRATIFICATION -

For this i have written a function IndexCreation first to divide the rows and columns by block number. No. of blocks are nothing but my total number of workers that I will be specifying while running my job. Index creation function has a generic logic for dividing the rows and columns.

[BELOW VALUES ARE ONLY FOR UNDERSTANDING. HAVE WORKED ON ACTUAL VALUES]

So e.g. if Blocks are 2 and my Matrix has 5(0, 1, 2, 3, 4) rows and 4(0, 1, 2, 3) columns, then I will have blocks with indices as below

Row index division into block - (0, 1) (2, 3, 4)

Column index division into block - (0, 1) (2, 3)

Then I take a Cartesian product of these 2 RDDs so that I get following index combinations

(0, 1)(0, 1) -- Matrix Block 1

(0, 1)(2, 3) -- Matrix Block 2

(2, 3, 4)(0, 1) -- Matrix Block 3

(2, 3, 4)(2, 3) -- Matrix Block 4

Now initialize W0 and H0 with random values between 0 and 1. W0 = Rows \* Factor and H0 = Factor \* Columns

Broadcasted w0 and h0

Declared and initialized variables that are required to calculate epsilon and broadcasted them

Now I create my Stratums in the function createStratumsVWH and stratumMaker

so I create block matrices from the original data matrix V as per the indices created in the function indexCreation.

so now,

STRATUM 1 will have diagonal blocks [0 0] [1 1] where 0 and 1 are pointers to the block indices

STRATUM 2 will have blocks [0 1] [1 0]

0 0 --> (0,1)(0,1)

1 1 --> (2,3,4)(2,3)

----------------------------

0 1 --> (0,1)(2,3)

1 0 --> (2,3,4)(0,1)

Using the above function I get my stratums with respective vij, wi\*, h\*j

3) ACTUAL ALGORITHM BEGINS

I have written a functions for SGD(createStratumsVWH) which has all the calculations as mentioned in the paper and assignment.

Then finally I begin Distributed SGD which runs for 100 iterations.

a) Create stratums

b) first work on diagonal stratums - send them to workers to do SGD for all nonzero values

c) update W and H and broadcast them

d) calculate variables for epsilon and broadcast the global parameter that will be used on worker to calculate epsilon

e) work on other stratums - send them to workers to do SGD for all nonzero values for every stratum respectively

f) update W and H and broadcast them

After 100 iterations, i get W and H updated.

4)L2 REGULARIZATION

a) get the original V matrix

b) get the new V by multiplying new W and H

c) take their difference

d) square each element and add them all. This is L(NZSL)

e) take individual W and H, square each element of W and H and add the squared values seperately for W and H. [W^2] + [H^2]

f) L2 = L(NZSL + lambda\*(SQRT(summed W^2) + SQRT(summed H^2)))

Q7) Spark implementation may be faster than the MapReduce version. Following might be the reasons -

1. Less IO overheads for Spark as compared to MapReduce since we have to work with iterations
2. Another reason may be, spark supports in memory data sharing across all its DAGs(directed acyclic graphs). Due to this different jobs using the same shared data can work efficiently at a higher speed.
3. Spark smartly uses RAM memory and Disk memory due to which it might be faster than MapReduce.
4. Concept of caching in Spark might prove helpful. In my case I had cached the Vij matrix because it was not going to be modified but just used for stratification. Things like these may help to get a better speed.