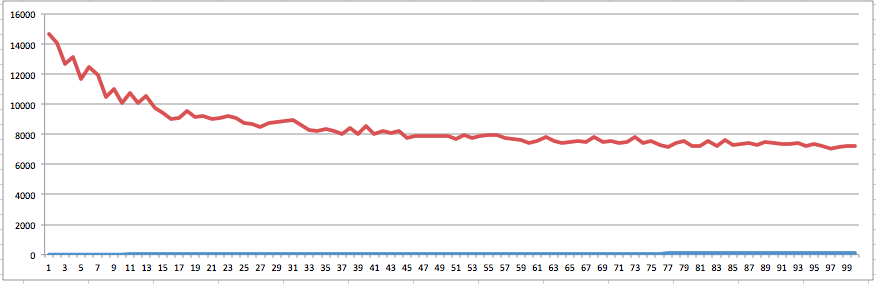
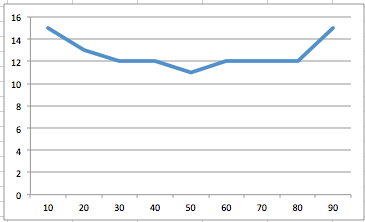
experiments:

1,



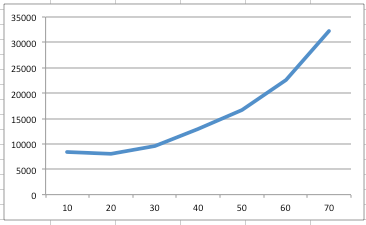
As we can see,as the iteration increases, the loss decreased.

2,



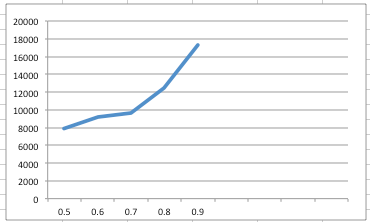
At the beginning, As the number of workers increase, the runtime will go down very fast. But if the number of workers reach to a point, then the runtime will not go down any more. And as the overhead of maintaining workers increasing, the runtime even increased.

3,



The number of factors is like the number of features. If it is too small, then it cannot represent the items very well. But if it’s too large, there will be too much noisy features, so the model cannot capture the real characteristics of the data. From the experiment, we can see that when F>10, as F increased, the loss increased. Thus, for this dataset, F should not set too large.

4,



As beta increased, the step size decreased. The result of this experiment is not what I expecteted. I expect that as the step size decrease, the speed of converge will be very slower, but the loss will decrease.

(1) Is there any advantage to using DSGD for Matrix Factorization instead of Singular Value Decomposition (SVD) which also finds a matrix decomposition that can be used in recommendation systems?

SVD has to computate the singular value first and it’s very complicated and time-consuming. So it’ll be slower than DSGD.

But both of them can use distributed method to do.

(2) Explain clearly and concisely your method (used in the code you have written) for creating strata at the beginning of every iteration of the DSGD-MF algorithm.

**Solution**

Suppose there are K strata and per straum there are K blocks.

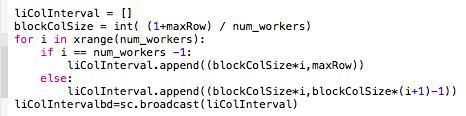
For each (i,j,r), I map to (i,[i,jr]). Then I use row number/blockRowSize as key.

The data becomes (key,[I,j,r])

Then I use partitionBy() to partition rows with the same key to a worker.



Also, I divide the columns to different blocks. Then in each worker,I can use this and the permutation sequence to choose blocks to do the SGD update.



In each iteration, I first create some permutations sequence.

permArr=np.random.permutation(num\_workers)

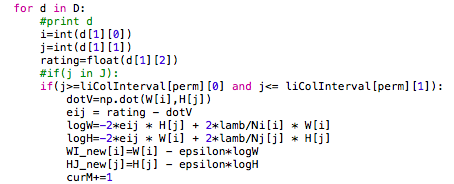
permbd=sc.broadcast(permArr)

Then I broadcast this to the worker.

I use the permutation sequence to choose the blocks in a stratum. Then in each worker, I just extract those blocks according to the permutation sequence.

permArr=permbd.value

perm=permArr[key]



(3) If you were to implement two versions of DSGD-MF using MapReduce and Spark, do you think you will find a relative speedup factor between MapReduce and Spark implementations.

**Solution**

In general, the Spark will be faster, because it supports in-memory computing. Using cache, the data can be keep in memory when doing iteration. So it don’t have to load the data over and over again.

4）

1-e

2-a

3-d

4-b

5-c

(5)

The link is

<https://github.com/lpxbetter/leetcode/blob/master/dsgb_mf.py>

README:

<https://github.com/lpxbetter/leetcode/blob/master/README.md>