10-605 - HW 5 - Distributed SGD for Matrix Factorization on Spark

Due: Thursday, November 12, 2015 23:59 EST (Autolab)
Late submission until: Saturday, November 14, 2015 23:59 EST (Autolab)
TAs: Kavya Srinet, Iosef Kaver, Ankit Agarwal

Name: ______________________________
Andrew ID: ___________________________

Guidelines for Answers: Please answer to the point. Please state any additional assumptions you make while answering the questions. You need to submit a single tar file on autolab, which should include your report. Please make sure you write the report legibly for grading.

Rules for Student Collaboration: The purpose of student collaboration in solving assignments is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is allowed to seek help from other students in understanding the material needed to solve a homework problem, provided no written notes are taken or shared during group discussions. The actual solutions must be written and implemented by each student alone, and the student should be ready to reproduce their solution upon request. You may ask clarifying questions on Piazza. However, under no circumstances should you reveal any part of the answer publicly on Piazza or any other public website. Any incidents of plagiarism or collaboration without full disclosure will be handled severely in accordance with CMU's Policy on Academic Integrity.

Rules for External Help: Some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions. Avoiding the use of heavily tested assignments detracts from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be available online or from other people. It is explicitly forbidden to use any such sources or to consult people who have solved these problems before. You must solve the homework assignments completely on your own. We will mostly rely on your wisdom and honor to follow this rule. However, if a violation is detected, it will be dealt with harshly.

Code of Conduct Declaration

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ______________________________ (e.g. *Jane explained to me what is asked in Question 3.4*)
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No.
- If you answered *yes*, give full details: ______________________________ (e.g. *I pointed Joe to section 2.3 to help him with Question 2*).

Important Note

You are expected to use Python and Spark for this assignment. You cannot use any libraries beyond those already provided in Python. You can use only the built-in constructs of PySpark and are not allowed to use `mllib` or any other Spark library. It could take hours to run and debug your experiments. Start early. **Note that HW6 might build on top of this homework, so make sure to give this homework an honest attempt.**

Kavya Srinet, Iosef Kaver and Ankit Agarwal are the contact TAs for this assignment. Please post clarification questions to Piazza, and the instructors can be reached at the following email address: `10605-instructors@cs.cmu.edu`.

Brief Introduction to Spark

Spark is a data science software that allows you to write your data processing code in Scala, Python, or Java. The data is loaded as an *Resilient Distributed Database (RDD)* from either the local filesystem or HDFS. RDDs can be converted into other RDDs using transformations such as map, filter, reduceByKey, etc. The evaluation of RDDs is lazy i.e. the required result won't be evaluated until you explicitly invoke an action indicating that you need the result. This allows Spark to optimize the execution of transformations scheduled on RDDs.

Another useful feature of Spark is in-memory processing. You can specify that you want to cache an RDD in memory if you intend to reuse the RDD through multiple iterations of your data processing job. The full set of *transformations* that convert one RDD into another, and *actions* which force the calculation of a result can be found in the Spark programming guide. The programming guide is also a good introduction to Spark. A more detailed RDD API reference with examples can be found here. If you prefer a lecture, you can try the tutorial from Spark Summit 2013 available here.

Spark on Andrew and AWS

Spark is placed in the `bigML` directory. You need to add the location of Spark binaries to your `PATH` variable to be able to run Spark on the Andrew cluster machines. Add the following line to your `.cshrc` file to access Spark on `unix.andrew.cmu.edu` machines:

```
setenv PATH ${PATH}:/afs/cs.cmu.edu/project/bigML/spark/bin
```

Algorithm 1 Stochastic Gradient Descent for Matrix Factorization

```
Require: Training indices $\mathbf{Z}$, Training data $\mathbf{V}$, randomly initialized $\mathbf{W}_0$ and $\mathbf{H}_0$
  while not converged do
    Select a training point $(i, j) \in \mathbf{Z}$ uniformly at random
    $\mathbf{W}'_{i*} \leftarrow \mathbf{W}_{i*} - \epsilon_n \frac{\partial}{\partial \mathbf{W}_{i*}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$
    $\mathbf{H}'_{*j} \leftarrow \mathbf{H}_{*j} - \epsilon_n \frac{\partial}{\partial \mathbf{H}_{*j}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j})$
    $\mathbf{W}_{i*} \leftarrow \mathbf{W}'_{i*}$
    $\mathbf{H}_{*j} \leftarrow \mathbf{H}'_{*j}$
  end while
```

Big Matrix Factorization using Spark

In this assignment, we will implement Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent (DSGD-MF) in Spark. The paper sets forth a solution for matrix factorization using minimization of sum of local losses. The solution involves dividing the matrix into strata for each iteration and performing sequential stochastic gradient descent within each stratum in parallel. The two losses considered are the plain non-zero square loss and the non-zero square loss with L_2 regularization of parameters W and H:-

$$L_{ij} = l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j}) = (\mathbf{V}_{ij} - \mathbf{W}_{i*}\mathbf{H}_{*j})^2 \tag{1}$$

$$L_{NZSL} = \sum_{(i,j)\in\mathbf{Z}} L_{ij} \tag{2}$$

Recall that DSGD-MF is a fully distributed algorithm i.e. both the data matrix $\mathbf{V}$ and factor matrices $\mathbf{W}$ and $\mathbf{H}$ can be carefully split and distributed to multiple workers for parallel computation without communication costs between the workers. Hence, it is a good match for implementation in a distributed in-memory data processing system like Spark. We outline the sequential algorithm and describe the steps needed to make it ready for distributed execution.

(a) Stochastic Gradient Descent for Matrix Factorization (SGD-MF)

In SGD-MF (1), we select a single datapoint and update the corresponding row of $\mathbf{W}$ and column of $\mathbf{H}$ in the direction of negative gradient.

The gradients for L_{NZSL} loss are given as follows:-

$$\frac{\partial}{\partial \mathbf{W}_{i*}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j}) = -2(\mathbf{V}_{ij} - \mathbf{W}_{i*}\mathbf{H}_{*j})\mathbf{H}_{*j} \tag{3}$$

$$\frac{\partial}{\partial \mathbf{H}_{*j}} l(\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j}) = -2(\mathbf{V}_{ij} - \mathbf{W}_{i*}\mathbf{H}_{*j})(\mathbf{W}_{i*})^T \tag{4}$$

(b) Stratified Stochastic Gradient Descent for Matrix Factorization (SSGD-MF)

Note that the calculation of gradient of local loss L_{ij} and its use in updating $\mathbf{W}$ and $\mathbf{H}$ is dependent on the entry $\mathbf{V}_{ij}$, the i^{th} row of $\mathbf{W}$: $\mathbf{W}_{i*}$, and the j^{th} column of $\mathbf{H}$: $\mathbf{H}_{*j}$.

A pair of elements of a matrix given as (i, j) and (i', j') is interchangeable if $i \neq i'$ and $j \neq j'$. See figure (1) for an example. If two such elements are interchangeable, then the stochastic gradient descent updates involving $\{\mathbf{V}_{ij}, \mathbf{W}_{i*}, \mathbf{H}_{*j}\}$ and $\{\mathbf{V}_{i'j'}, \mathbf{W}_{i'*}, \mathbf{H}_{*j'}\}$ do not depend on each other in any way and can be performed in parallel.

A set of elements of a matrix is interchangeable if any pair of elements in the set is interchangeable. The stochastic gradient descent updates involving the local losses of a set of interchangeable elements are also parallel due to the fact that they depend on disjoint parts of the data matrix $\mathbf{V}$ and the factor matrices $\mathbf{W}$ and $\mathbf{H}$.

We can generalize the notion of interchangeability from elements of a matrix to blocks of the matrix. Consider I and I' are sets of row indices of $\mathbf{V}$. Similarly, J and J' are sets of column indices of $\mathbf{V}$. Matrix blocks IJ and $I'J'$ are interchangeable if $I \cap I' = \phi$ and $J \cap J' = \phi$, where IJ denotes the cartesian product of sets I and J, and $\mathbf{V}_{IJ}$ denotes a matrix block with some abuse and inelegance of notation. Similar to the case of element exchangeability, the stochastic gradient descent updates involving $\{\mathbf{V}_{IJ}, \mathbf{W}_{I*}, \mathbf{H}_{*J}\}$ and $\{\mathbf{V}_{I'J'}, \mathbf{W}_{I'*}, \mathbf{H}_{*J'}\}$ do not depend on each other in any way and can be performed in parallel.

As in the case of element interchangeability, block interchangeability generalizes from a pair of matrix blocks to a set of matrix blocks. A set of matrix blocks is said to be interchangeable if any pair of those matrix blocks is interchangeable. Stochastic gradient descent updates involving interchangeable matrix blocks can be parallelized since they depend on disjoint parts of the data matrix $\mathbf{V}$ and the factor matrices $\mathbf{W}$ and $\mathbf{H}$.

Given a matrix block $\mathbf{V}_{IJ}$ and the coupled parameter blocks $\mathbf{W}_{I*}$ and $\mathbf{H}_{*J}$, we can perform sequential stochastic gradient descent for this disjoint part of the matrix and its parameters without having to worry about any parallel updates that might affect $\mathbf{W}_{I*}$ and $\mathbf{H}_{*J}$. Thus, we have established a concurrent model for stochstic gradient descent updates for matrix factorization. Interchangeable matrix blocks are also called *strata*, and hence this concurrent algorithm of performing SGD is also called Stratified Stochastic Gradient Descent (SSGD).

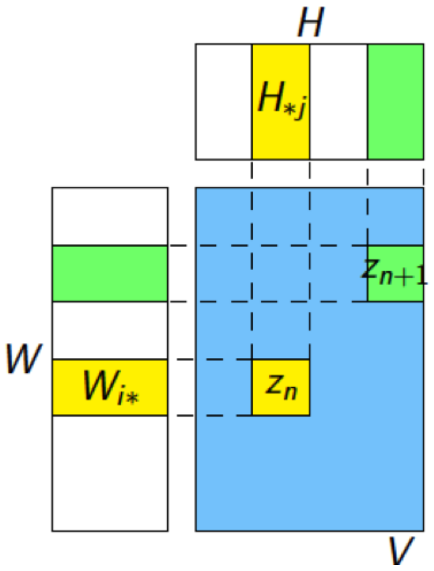

Figure 1: Element Interchangeability in $\mathbf{V}$

(c) Distributed Stochastic Gradient Descent for Matrix Factorization (DSGD-MF)

Having established concurrency in SSGD-MF, here we describe how to parallelize SSGD to yield Distributed Stochastic Gradient Descent algorithm for matrix factorization (DSGD-MF). The algorithm is given in (2). As long as the parameter estimates $\mathbf{W}$ and $\mathbf{H}$ have not converged, we choose a set of strata, which creates disjoint blocks of $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{H}$ that can be handed over to workers for performing sequential stochastic gradient descent. Each worker performs SGD updates for the block of parameters it has been handed and returns the updated parameter blocks. For convenience, during iteration i, we will allow the n^{th} worker to perform m_{ni} updates where m_{ni} is the number of non-zero entries in the data matrix block $V_{I_n J_n}$ handed to that worker during iteration i. The worker will sequentially go through all the non-zero entries in its block $V_{I_n J_n}$ and perform an SGD update for each entry as explained in SGD-MF earlier.

Algorithm 2 Distributed Stochastic Gradient Descent for Matrix Factorization

```
Require: Training data $\mathbf{V}$, randomly initialized $\mathbf{W}_0$ and $\mathbf{H}_0$, the number of parallel workers $B$, the number of factors $F$
  Partition V matrix across $B$ workers.
  Blockify the matrix and assign the correct block to each $V_{ij}$
  for $curr\_iteration \in 1..Total\_iterations$ do
    Broadcast W and H matrices
    Filter blocks for the current strata
    Perform SGD updates for W and H as explained in Algorithm 1
    Update W and H by collecting the updated blocks from all workers and aggregating the updates
  end for
```

Experiments

We will be using the RCV1 small dataset for the experimental evaluation. This is a very popular dataset that has news documents that have been tagged to certain classes. For your convenience, we have preprocessed the dataset. The dataset is available at `/afs/cs.cmu.edu/project/bigML/RCV1-MF/RCV1.small_train.txt`. You will perform your experimental evaluations on the available dataset. It is in the triples format:

```
<word_1>,<doc_1>,<tfidf_11>
...
<word_j>,<doc_i>,<tfidf_ij>
...
<word_N>,<doc_M>,<tfidf_ij>
```

Here, `doc_i` is an integer ID for the i^{th} document, `word_j` is an integer ID for the j^{th} word in the Vocabulary, and `tfidf_ij` is the tf-idf score for `word_j` in `document_i`.

For example, the content of the input file to your script will look like:-

```
3114,1,1.25
608,1,2.60
1246,1,0.15
1357,2,4.125
3068,2,2.30
1537,2,1.575
...
562,6040,0.45
1096,6040,5.40
1097,6040,.15
```

We have also given you the file : `/afs/cs.cmu.edu/project/bigML/RCV1-MF/RCV1.small_docIds.txt`. The data in this file contains the document IDs for each document, so the data looks something like:-

```
<doc_1> <doc-content_1>
```

```
...
<doc_i> <doc-content_i>
...
<doc_M> <doc-content_M>
```

Here, `doc_i` is an integer ID for the i^{th} document and `doc-content_i` is the content of i^{th} document from the RCV1 small dataset. The ID and content are tab-separated.

We have given you another file : `/afs/cs.cmu.edu/project/bigML/RCV1-MF/RCV1.small_idf.txt` This file contains the mapping of a word to its numeric ID. The content of the file looks something like:-

```
<word_1>    <wordId_1>  <wordIdf_1>
...
<word_j>    <wordId_j>  <wordIdf_j>
...
<word_N>    <wordId_N>  <wordIdf_N>
```

Here, `word_j` is the word, the `wordId_j` is an integer ID for the j^{th} word and `wordIdf_j` is the IDF score for the word in the given RCV1_small corpus. These three are also tab-separated.

We will explore various choices for the number of iterations I, the number of workers B, the number of factors F, and the inner SGD step size ϵ_n. The SGD stepsize should be set as $\epsilon_n = (\tau_0 + n)^{-\beta}$ where n is the iteration number, so that it decays with iteration number. We will set $\tau_0 = 100$ and vary ϵ_n by varying β.

Note that you will not have access to the exact iteration number n to calculate ϵ_n while performing SGD updates within a stratum, since strata-specific SGDs are performed in parallel. You need to use an approximate version of the iteration number to calculate ϵ_n. In particular, you should use $n = n' + \sum_{i,b} m_{bi}$ where $\sum_{i,b} m_{bi}$ is the total number of SGD updates made across all strata in all previous iterations and n' is the number of SGD updates made by the current worker on its stratum so far. Thus, ϵ_n is synchronized for all workers at the end of every iteration, but is allowed to be calculated in a decoupled fashion once a worker starts performing SGD updates for the current iteration on its local stratum.

Also, remember to randomly initialize your factor matrices $\mathbf{W}$ and $\mathbf{H}$; do not initialize them to zero matrices.

Deliverables

In this assignment, you are required to required to implement DSGD-MF in Python using PySpark. We have provided you with starter code that can be downloaded from the "Download handout" section of the Homework5: Matrix Factorization link. You need to test code on the RCV1.small_train.txt, provide plots and explanations for your experiments, and provide answers to the theory questions asked in this assignment. You need to submit your implementations via Autolab where it will be judged on the matrix reconstruction error, time efficiency, and memory efficiency. You should implement the algorithm by yourself instead of using any existing machine learning toolkit. You should upload your code (including all your function files) along with a report named `andrewid-report.pdf`, which should answer the following questions in the space provided:

(1) Is there any advantage to using DSGD for Matrix Factorization instead of Singular Value Decomposition (SVD) which also finds a matrix decomposition?

[10 points]

(2) Answer the questions below:

- What do W and H matrices represent in the context of topic modelling using DSGD? An intuitive explanation should suffice.

  [5 points]

- How would you go about determining the top k words in a topic?

  [5 points]

(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, keeping other parameters like the total number of iterations and number of workers fixed? Which implementation do you think will be faster? Why? If your answer depends on any general optimization tricks related to MapReduce or Spark that you know, please state them as well.

[10 points]

(4) Please perform the following experiments and attach your plots and answers in the report:-

- Set the number of workers $B = 10$, the number of factors $F = 20$, and $\beta = 0.8$. Plot the reconstruction error L_{NZSL} versus the iteration number $i = 1, 2, .., 100$. Explain the trend in your plot in the space provided below.

  [*10 points*]

- Set the number of iterations $I = 30$, the number of factors $F = 20$, and $\beta = 0.8$. Plot the runtime of your Spark code R versus number of workers $B = 2, 3, .., 10$ in steps of 1. Please ensure your local machine or Spark cluster can support the number of parallel workers you are requesting. Explain the trend in your plot in the space provided below.

  [*10 points*]

- Set the number of iterations $I = 50$, the number of workers $B = 10$, and $\beta = 0.95$. Plot the reconstruction error L_{NZSL} versus number of factors $F = 10, 20, .., 80$ in steps of 10. Explain the trend in your plot in the space provided below.

  [*10 points*]

- Set the number of workers $B = 10$, the number of factors $F = 20$, and the number of iterations $I = 30$. Plot the reconstruction error L_{NZSL} versus $\beta = 0.5, 0.6, .., 0.9$ in steps of 0.1. Explain the trend in your plot in the space provided below.

  [*10 points*]

(5) Answer the questions in the collaboration policy on page 1.

Autolab Submission and Evaluation Details

Your DSGD-MF program should be named `dsgd_mf.py` should be able to run without any issues using the following command:

```
spark-submit --driver-memory 2G dsgd_mf.py <num_factors> <num_workers> <num_iterations> \
        <beta_value> \
        <inputV_filepath> <outputW_filepath> <outputH_filepath>
```

For example,

```
spark-submit --driver-memory 2G dsgd_mf.py 100 10 50 0.95 test.csv w.csv h.csv
```

Factor matrices obtained from matrix decompositions are often dense and either tall or fat i.e. one of the dimensions is pretty small. Hence, the comma-delimited format is not efficient for storing factor matrices. Your factor matrices should be written to the files `w.csv` and `h.csv` (provided as commandline arguments) in the normal, non-sparse CSV format. For example, see the following `4x5` matrix.

```
3.0,6,9,12,15
5,10,15.2,20,25
7,14,21.5,28,35
11,22.2,33,44.8,55
```

You should tar the following items into **hw5.tar** and submit to the homework5 assignment link via Autolab:

- `andrewid-report.pdf`
- `dsgd_mf.py`,and all other auxiliary files required by `dsgd_mf.py`
- `w.csv` and `h.csv` output files from running your code on `RCV1.small_train.txt`.
- Spark logfile named `spark_dsgd.log` from running your code on `RCV1.small_train.txt`. The logfile should have all messages printed to terminal when Spark is run on `RCV1.small_train.txt` using the following command:

  ```
  spark-submit dsgd_mf.py 30 3 30 0.9
           \ RCV1.small_train.txt w.csv h.csv
  ```

Tar the files directly using `tar -cvf hw5.tar *.py *.csv *.log andrewid-report.pdf`. Do **NOT** put the above files in a folder and then tar the folder. You do not need to upload any saved temporary files. Please make sure your code is working fine before you submit.

You must submit your homework through Autolab via the Homework5: Matrix Factorization link. You have a total of **10 unpenalized submissions** after which each additional submission will be penalized by an additional 10 percent. Your performance will be evaluated, and grades from automated evaluation will be provided immediately.

[*30 points*]

Grading

The total grade of this assignment is **100 points**.

- Tests on your autolab code carry **30 points**. You will be graded based on the memory usage (10 points) and runtime (10 points) for your code, and your final reconstruction error (10 points).
- Experimental results (plots and explanations) on the preprocessed RCV1.small dataset: **40 points**.
- The theory questions in the assignment will be graded manually: **30 points**.

$$\boxed{\textbf{Total: 100}}$$