INF 553 – Spring 2017

Homework #4: Collaborative Filtering

Due: March 24, Friday
100 points

In this homework, we consider the latent factor modeling of utility matrix M (e.g., a rating matrix where rows represent users and columns products such as movies). Recall that in latent factor modeling, the goal is to decompose M into lower-rank matrices U and V such that the difference between M and UV is minimized.

Tasks

The homework consists of the following **two** tasks:

1. [80 points] Implement the incremental UV decomposition algorithm as discussed in class (and described in the text), where the element is learned one at a time.

   Assume initially all elements in latent factor matrix U and V are 1's.

   The learning starts with learning elements in U **row by row**, i.e., U[1,1], U[1,2], …, U[2, 1], …
   It then moves on to learn elements in V **column by column**, i.e., V[1,1], V[2,1], …, V[1, 2], …
   When learning an element, it uses the latest value learned for all other elements. It should compute the optimal value for the element to minimize the current RMSE as described in class.

   The learning process stops after a specified number of iterations, where a round of learning all elements in U and V is one iteration.

   The algorithm should output RMSE after each iteration (remember that the mean is computed based on non-blank elements in the input matrix M).

   Write your code in Python (**DO NOT** use Spark here). Name your algorithm, <FirstName>_<LastName>_ uv.py. It can be invoked as follows.

   <u>**Execution format:**</u>
   python uv.py input-matrix n m f k

   - "input-matrix" is a utility matrix described above. It comes in **sparse format**. For example, the matrix you saw in class can be represented as follows.
     ```
     1,1,5
     1,2,2
     1,3,4
     1,4,4
     1,5,3
     2,1,3
     ```

…

- n is the number of rows (users) of the matrix, while m is the number of columns (products).
- f is the number of dimensions/factors in the factor model. That is, U is n-by-f matrix, while V is f-by-m matrix.
- k is the number of iterations.

For example, python john_smith_uv.py matrix.dat 5 5 2 10.

__Output format:__
In this task, **DO NOT** write the output to any files. Use **standard output** to print them instead.
After each iteration, output RMSE with 4 floating points, so you could use ***print "%.4f" % RMSE*** to print results as follows.

```
1.0019
0.9794
0.8464
…
```

__Submission__:
Submit a Python script in the form: john_smith_uv.py, that is your first and last names followed by the name of program.

2. [20 points] In this task, you are asked to modify the parallel implementation of ALS (alternating least squares) algorithm in Spark, so that it takes a utility matrix as the input, and output the RMSE into a file after each iteration.
The code for the algorithm is als.py under the <your spark-2.1.0 installation directory>/examples/src/main/python. A copy is also provided with this handout.
Note that make sure that you use the version for **spark-2.1.0** (it is slightly different from previous verions).

__Things to do:__
1. Take a utility matrix as the input
2. Output RMSE into a file after each iteration

__Execution format:__
```
bin/spark-submit als.py input-matrix n m f k p output-file
```

All parameters are the same as for uv.py, except for an additional parameter p, which is the number of partitions for the input-matrix, and an additional parameter output-file, which is the path to the output file.

Output format:

In this task, write the output to a file.

After each iteration, output RMSE with 4 floating points, so you could use ***"%.4f" % RMSE*** to format the RMSE value, and save into file as follows.

```
1.0019
0.9794
0.8464
…
```

Submission: <FirstName>_<LastName>_ als.py

Notes:

- You may use numpy package in this homework. To install it on EC2, execute “sudo pip install numpy”.
- **We will be using Moss for plagiarism detection**. **Do not copy from each other or you will face serious consequences!**