CS246: Mining Massive Data Sets **Winter 2016**

Problem Set 3

Due 11:59pm February 18, 2016

Only one late period is allowed for this homework (11:59pm 2/23).

General Instructions

Submission instructions: These questions require thought but do not require long answers. Please be as concise as possible. You should submit your answers as a writeup in PDF format via GradeScope and code via the Snap submission site.

Submitting writeup: Prepare answers to the homework questions into a single PDF file and submit it via http://gradescope.com. Make sure that the answer to each question is on a *separate page*. On top of each page write the number of the question you are answering. Please find the cover sheet and the recommended templates located here:
http://web.stanford.edu/class/cs246/homeworks/hw3/submission_template_hw3.tex
http://web.stanford.edu/class/cs246/homeworks/hw3/submission_template_hw3.pdf
http://web.stanford.edu/class/cs246/homeworks/hw3/submission_template_hw3.docx

Not including the cover sheet in your submission will result in a 2-point penalty. It is also important to tag your answers correctly on Gradescope. We will deduct 5/N points for each incorrectly tagged subproblem (where N is the number of subproblems). This means you can lose up to 5 points for incorrect tagging.

Submitting code: Upload your code at http://snap.stanford.edu/submit. Put all the code for a single question into a single file and upload it.

Questions

1 Latent Features for Recommendations (25 points)

Warning: This problem requires substantial computing time (it can be a few hours on some systems). Don't start it at the last minute.

* * *

The goal of this problem is to implement the *Stochastic Gradient Descent* algorithm to build a Latent Factor Recommendation system. We can use it to recommend movies to users.

We encourage you to read the slides of the lecture "Recommender Systems 2" again before attempting the problem.

Suppose we are given a matrix R of recommendations. The element R_{iu} of this matrix corresponds to the rating given by user u to item i. The size of R is $m \times n$, where m is the number of movies, and n the numbers of users.

Most of the elements of the matrix are unknown because each user can only rate a few movies.

Our goal is to find two matrices P and Q, such that $R \simeq QP^T$. The dimensions of Q are $m \times k$, and the dimensions of P are $n \times k$. k is a parameter of the algorithm.

We define the error as

$$E = \sum_{(i,u)\in \text{ratings}} (R_{iu} - q_i \cdot p_u^T)^2 + \lambda \left[\sum_u \|p_u\|_2^2 + \sum_i \|q_i\|_2^2 \right]. \quad (1)$$

The $\sum_{(i,u)\in \text{ratings}}$ means that we sum only on the pairs (user, item) for which the user has rated the item, *i.e.* the (i, u) entry of the matrix R is known. q_i denotes the i^{th} row of the matrix Q (corresponding to an item), and p_u the u^{th} row of the matrix P (corresponding to a user u). λ is the regularization parameter. $\|\cdot\|_2$ is the L_2 norm and $\|p_u\|_2^2$ is square of the L_2 norm, *i.e.,* it is the sum of squares of elements of p_u.

(a) [5 points]

Let ε_{iu} denote the derivative of the error E with respect to R_{iu}. What is the expression for ε_{iu}? What are the update equations for q_i and p_u in the Stochastic Gradient Decent algorithm?

(Hint: See lecture slides.)

(b) [20 points]

Implement the algorithm. Read each entry of the matrix R from disk and update ε_{iu}, q_i and p_u for each entry.

To emphasize, you are not allowed to store the matrix R in memory. You have to read each element R_{iu} one at a time from disk and apply your update equations (to each element). Then, iterate until both q_i and p_u stop changing. Each iteration of the algorithm will read the whole file.

Choose $k = 20$, $\lambda = 0.2$ and number of iterations $= 40$. Find a good value for the learning rate η. Start with $\eta = 0.1$. The error E on the training set ratings.training.txt discussed below should be less than 83000 after 40 iterations.

Based on values of η, you may encounter the following cases:

- If η is too big, the error function can converge to a high value or may not monotonically decrease. It can even diverge and make the components of vectors p and q equal to ∞.
- If η is too small, the error function doesn't have time to significantly decrease and reach convergence. So, it can monotonically decrease but not converge *i.e.* it could have a high value after 40 iterations because it has not converged yet.

Use the dataset at http://snap.stanford.edu/class/cs246-data/hw3-recommendations.zip. It contains the following files:

- `ratings.train.txt`: This is the matrix R. Each entry is made of a user id, a movie id, and a rating.
- `ratings.val.txt`: This is the test set. You will use it to evaluate your recommendation system. It consists of entries of the matrix that were removed from the original dataset to create the training set.

Plot the value of the objective function E (defined in equation 1) on the training set as a function of the number of iterations. What value of η did you find?

You can use any programming language to implement this part, but Java, C/C++, and Python are recommended for speed. (In particular, Matlab can be rather slow reading from disk.) It should be possible to get a solution that takes on the order of minutes to run with these languages.

Hint: These hints will help you if you are not sure about how to proceed for certain steps of the algorithm, although you don't have to follow them if you have another method.

- *Determine the dimensions of P and Q. You can compute the maximal userID and movieID from a pass through the data. (You should not assume these constants are known at the start of your program.) This allows you to store P and Q in "sparse" matrices; for items i and users u not present in the training set, the rows q_i and p_u will never be updated.*
- *Initialization of P and Q: We would like q_i and p_u for all users u and items i such that $q_i \cdot p_u^T \in [0, 5]$. A good way to achieve that is to initialize all elements of P and Q to random values in $[0, \sqrt{5/k}]$.*
- *Update the equations: In each update, we update q_i using p_u and p_u using q_i. Compute the new values for q_i and p_u using the old values, and then update the vectors q_i and p_u.*
- *You should compute E at the end of a full iteration of training. Computing E in pieces during the iteration is incorrect since P and Q are still being updated.*

What to submit

(i) Equation for ε_{iu}. Update equations in the Stochastic Gradient Descent algorithm. [1(a)]

(ii) Value of η. Plot of E vs. number of iterations. Make sure your graph has a y-axis so that we can read the value of E. [1(b)]

(iii) Please upload all the code to snap submission site. [1(b)]

2 Implementing PageRank and HITS (25 points)

In this problem, you will learn how to implement the PageRank and HITS algorithms. You will be experimenting with a small randomly generated graph (assume graph has no dead-ends) provided at http://snap.stanford.edu/class/cs246-data/graph.txt.

It has $n = 100$ nodes (numbered $1, 2, \ldots, 100$), and $m = 1024$ edges, 100 of which form a directed cycle (through all the nodes) which ensures that the graph is connected. It is easy to see that the existence of such a cycle ensures that there are no dead ends in the graph. There may be multiple edges between a pair of nodes, your program should handle these instead of ignoring them. The first column in `graph.txt` refers to the source node, and the second column refers to the destination node.

(a) PageRank Implementation [12 points]

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, \ldots, n$) and m edges, all nodes have positive out-degree, and $M = [M_{ji}]_{n\times n}$ is a an $n \times n$ matrix as defined in class such that for any $i, j \in [\![1, n]\!]$:

$$M_{ji} = \begin{cases} \frac{1}{\deg(i)} & \text{if } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $\deg(i)$ is the number of outgoing edges of node i in G. By the definition of PageRank, assuming $1 - \beta$ to be the teleport probability, and denoting the PageRank vector by the column vector r, we have the following equation:

$$r = \frac{1-\beta}{n}\mathbf{1} + \beta M r, \tag{2}$$

where $\mathbf{1}$ is the $n \times 1$ vector with all entries equal to 1.

Based on this equation, the iterative procedure to compute PageRank works as follows:

1. Initialize: $r^{(0)} = \frac{1}{n}\mathbf{1}$

2. For i from 1 to k, iterate: $r^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M r^{(i-1)}$

Run the aforementioned iterative process for 40 iterations and obtain the PageRank vector r. Compute the following:

- List the top 5 node ids with the highest PageRank scores.
- List the bottom 5 node ids with the lowest PageRank scores.

(b) HITS Implementation [13 points]

Assume the directed graph $G = (V, E)$ has n nodes (numbered $1, 2, \ldots, n$) and m edges, all nodes have non-negative out-degree, and $L = [L_{ij}]_{n\times n}$ is a an $n \times n$ matrix referred to as the *link matrix* such that for any $i, j \in [\![1, n]\!]$:

$$L_{ij} = \begin{cases} 1 & \text{if } (i \to j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

Given the link matrix L and some scaling factors λ, μ, the hubbiness vector h and the authority vector a can be expressed using the equations:

$$h = \lambda L a, a = \mu L^T h \tag{3}$$

where $\mathbf{1}$ is the $n \times 1$ vector with all entries equal to 1.

Based on this equation, the iterative method to compute h and a is as follows:

1. Initialize h with a column vector (of size $n \times 1$) of all 1's.
2. Compute $a = L^T h$ and scale so that the largest value in the vector a has value 1.
3. Compute $h = La$ and scale so that the largest value in the vector h has value 1.
4. Go to step 2.

Repeat the iterative process for 40 iterations, assume that $\lambda = 1, \mu = 1$ and then obtain the hubbiness and authority scores of all the nodes (pages). Compute the following:

- List the 5 node ids with the highest hubbiness score.
- List the 5 node ids with the lowest hubbiness score.
- List the 5 node ids with the highest authority score.
- List the 5 node ids with the lowest authority score.

What to submit

(i) List 5 node ids with the highest and least PageRank scores [2(a)]

(ii) List 5 node ids with the highest and least hubbiness and authority scores [2(b)]

(iii) Upload all the code to the snap submission site [2(a) & 2(b)]

3 Dead ends in PageRank computations (20 points)

Let the *matrix of the Web* M be an n-by-n matrix, where n is the number of Web pages. The entry m_{ij} in row i and column j is 0, unless there is an arc from node (page) j to node i. In that case, the value of m_{ij} is $1/k$, where k is the number of arcs (links) out of node j. Notice that if node j has $k > 0$ arcs out, then column j has k values of $1/k$ and the rest 0's. If node j is a *dead end* (i.e., it has zero arcs out), then column j is all 0's.

Let $\mathbf{r} = [r_1, r_2, \ldots, r_n]^T$ be (an estimate of) the PageRank vector; that is, r_i is the estimate of the PageRank of node i. Define $w(\mathbf{r})$ to be the sum of the components of $\mathbf{r}$; that is $w(\mathbf{r}) = \sum_{i=1}^{n} r_i$.

In one iteration of the PageRank algorithm, we compute the next estimate $\mathbf{r}'$ of the PageRank as: $\mathbf{r}' = M\mathbf{r}$. Specifically, for each i we compute $r'_i = \sum_{j=1}^{n} M_{ij} r_j$.

(a) [6pts]

Suppose the Web has no dead ends. Prove that $w(\mathbf{r}') = w(\mathbf{r})$.

(b) [7pts]

Suppose there are still no dead ends, but we use a teleportation probability of $1 - \beta$, where $0 < \beta < 1$. The expression for the next estimate of r_i becomes $r'_i = \beta \sum_{j=1}^{n} M_{ij} r_j + (1-\beta)/n$. Under what circumstances will $w(\mathbf{r}') = w(\mathbf{r})$? Prove your conclusion.

(c) [7pts]

Now, let us assume a teleportation probability of $1 - \beta$ in addition to the fact that there are one or more dead ends. Call a node "dead" if it is a dead end and "live" if not. Assume $w(\mathbf{r}) = 1$. At each iteration, we will distribute equally to each node the sum of:

1. $(1 - \beta) r_j$ if node j is live.
2. r_j if node j is dead.

Write the equation for r'_i in terms of β, M, and $\mathbf{r}$. Then, prove that $w(\mathbf{r}')$ is also 1.

What to submit

(i) Proof [3(a)]

(ii) Condition for $w(\mathbf{r}') = w(\mathbf{r})$ and Proof [3(b)]

(iii) Equation for r'_i and Proof [3(c)]

4 Dense Communities in Networks (30 points)

In this problem, we study the problem of finding dense communities in networks.

Definitions: Assume $G = (V, E)$ is an undirected graph (e.g., representing a social network).

- For any subset $S \subseteq V$, we let the *induced edge set* (denoted by $E[S]$) to be the set of edges both of whose endpoints belong to S.
- For any $v \in S$, we let $\deg_S(v) = |\{u \in S | (u, v) \in E\}|$.
- Then, we define the *density* of S to be:

$$\rho(S) = \frac{|E[S]|}{|S|}.$$

- Finally, the *maximum density* of the graph G is the density of the densest induced subgraph of G, defined as:

$$\rho^*(G) = \max_{S \subseteq V} \{\rho(S)\}.$$

Goal. Our goal is to find an induced subgraph of G whose density is not much smaller than $\rho^*(G)$. Such a set is very densely connected, and hence may indicate a community in the network represented by G. Also, since the graphs of interest are usually very large in practice, we would like the algorithm to be highly scalable. We consider the following algorithm:

Require: $G = (V, E)$ and $\epsilon > 0$
  $\tilde{S}, S \leftarrow V$
  while $S \neq \emptyset$ **do**
    $A(S) := \{i \in S \mid \deg_S(i) \leq 2(1 + \epsilon)\rho(S)\}$
    $S \leftarrow S \setminus A(S)$
    if $\rho(S) > \rho(\tilde{S})$ **then**
      $\tilde{S} \leftarrow S$
    end if
  end while

return $\tilde{S}$

The basic idea in the algorithm is that removing nodes with low degrees do not contribute much to the density of a dense subgraph, hence they can be removed without significantly decreasing the density. In fact, in some cases, their removal results in an increase in the density.

We analyze the quality and performance of this algorithm. We start with analyzing its performance.

(a) [12 points]

We show through the following steps that the algorithm terminates in a logarithmic number of steps.

i. Prove that at any iteration of the algorithm, $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$.

ii. Prove that the algorithm terminates in at most $\log_{1+\epsilon}(n)$ iterations, where n is the initial number of nodes.

(b) [18 points]

We show through the following steps that the density of the set returned by the algorithm is at most a factor $2(1+\epsilon)$ smaller than $\rho^*(G)$.

i. Assume S^* is the densest subgraph of G. Prove that for any $v \in S^*$, we have: $\deg_{S^*}(v) \geq \rho^*(G)$.

ii. Consider the first iteration of the while loop in which there exists a node $v \in S^* \cap A(S)$. Prove that $2(1+\epsilon)\rho(S) \geq \rho^*(G)$.

iii. Conclude that $\rho(\tilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$.

What to submit

(a) i. Proof of $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$.
ii. Proof of number of iterations for algorithm to terminate.

(b) i. Proof of $\deg_{S^*}(v) \geq \rho^*(G)$.
ii. Proof of $2(1+\epsilon)\rho(S) \geq \rho^*(G)$.
iii. Conclude that $\rho(\tilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$.