CS246: Mining Massive Data Sets **Winter 2015**

Problem Set 3

Due 5:00pm February 19, 2015

Only one late period is allowed for this homework (2/24).

General Instructions

These questions require thought, but do not require long answers. Please be as concise as possible. Please fill the cover sheet and submit it as a front page with your answers. We will **subtract 2 points** for failing to include the cover sheet. Include the names of your collaborators (see the course website for the collaboration or honor code policy).

All students (Regular and SCPD) should submit their answers using Scoryst (see course website FAQ for further instructions). This submission should include the source code you used for the programming questions.

Additionally, all students (Regular and SCPD) should upload all code through a single text file per question using the url below. Please do not use .pdf, .zip or .doc files for your code upload.

Cover Sheet: http://cs246.stanford.edu/cover.pdf

Code (Snap) Upload: http://snap.stanford.edu/submit/

Assignment (Scoryst) Upload: https://scoryst.com/course/39/submit/

Questions

1 Latent Features for Recommendations (30 + 5 points) [Clement, Jean-Yves]

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 $(\text{user}, \text{item})$ 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) [3 points]

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

(Hint: See lecture slides.)

(b) [15 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} 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 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.*

(c) [12 points]

We define the training error as:

$$E_{\text{tr}} = \sum_{(i,u) \in \text{ratings}} (R_{iu} - q_i \cdot p_u^T)^2,$$

and the test error as:

$$E_{\text{te}} = \sum_{(i,u)\in\text{test ratings}} (R_{iu} - q_i \cdot p_u^T)^2.$$

Note that these equations do not include the regularization penalty that E had from parts a and b.

'Test ratings' is the set of ratings given in the test set. If there are ratings (i, u) in the test set such that i or u is not in the training set, then you can choose to either include or exclude them from the sum, whichever is more convenient; your choice will not affect E_{te} significantly.

Plot E_{te} and E_{tr} as a function of $k \in \{1, 2, \ldots 10\}$ for the following values of λ.

- $\lambda = 0.0$
- $\lambda = 0.2$

For η use the value found in 1(b).

Based on your output, which of the following statements are valid ?

- A: Regularization increases the test error for $k \geqslant 5$
- B: Regularization decreases the test error for $k \geqslant 5$
- C: Regularization has no effect on the test error for $k \geqslant 5$
- D: Regularization increases the training error for all (or almost all) k
- E: Regularization decreases the training error for all (or almost all) k
- F: Regularization has no effect on the training error for all (or almost all) k
- G: Regularization increases overfitting
- H: Regularization decreases overfitting
- I: Regularization has no effect on overfitting

(d) [5 bonus points]

Another model presented in class is defined by:

$$R_{iu} = \mu + b_u + b_i + q_i \cdot p_u^T,$$

where μ is the overall mean rating and b_u is the bias for user u , b_i is the bias for item i. b_u and b_i are learnt in this model.

Using the new error function defined on slide 44 in the lecture on "Recommender systems: Latent Factor Models (Recommender Systems 2)", compute the gradient descent rules. Apply stochastic gradient descent to train this model.

Answer parts (a), (b) and (c) for this new model.

Note: The bias parameters do not scale the way the elements of p and q do. You may need to use multiple η's (one for p and q, and one for the bias parameters) to get decent results. If you do, report your choices.

What to submit

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

(b) 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. Code: Scoryst and Snap.

(c) Plots of E_{te} vs. k and E_{tr} vs. k for the two values of λ. A list of valid statements. Make sure your graphs have a y-axis so that we can read the value of the E_{te} and E_{tr}. Code: Scoryst and Snap.

(d) Equations. Value of η. Plot of E vs. number of iterations. Plots of E_{te} vs. k and E_{tr} vs. k for the two values of λ. A list of valid statements. Make sure your graphs have a y-axis so that we can read the value of error. Code: Scoryst and Snap.

2 PageRank Computation (30 points) [Hristo, Peter, James]

In this problem, we study the Power Iteration method and a variant of the Monte Carlo approach for estimating the PageRank vector. 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 Power Iteration method works as follows:

1. Initialize: $r^{(0)} = \frac{1}{n}\mathbf{1}$
2. For i from 1 to k, iterate: $r^{(i)} = \frac{1-\beta}{n}\mathbf{1} + \beta M r^{(i-1)}$

(a) [7pts]

Prove that:

$$||r - r^{(k)}||_1 \leq 2\beta^k,$$

where $||\cdot||_1$ denotes the L_1 norm.

(b) [4pts]

Show that to guarantee the maximum L_1 error from part a) to be less than a constant δ, the running time of the Power Iteration method is $O(\frac{m}{\log(1/\beta)})$, where m is the number of edges in the graph. *Hint: Start by trying to find a bound for the number of iterations in order to obtain a max L_1 error less than constant δ.*

* * *

This wraps up the basic analysis of the Power Iteration method. Now, for a variant of the Monte Carlo approach for approximation of PageRank, we consider the following algorithm, which we call the *MC algorithm*:

MC Algorithm: Consider a random walk $\{X_t\}_{t\geq 0}$ that starts from a certain node, and at each step terminates with probability $1-\beta$, or follows a uniformly random outgoing edge (from the current node) with probability β. Simulate this random walk exactly R times from each node (total of $N = nR$ runs). Evaluate $\widetilde{r_j}$ (defined for all $j \in [\![1; n]\!]$) as the total number of visits to the node j in all steps of all these walks, divided by $\frac{nR}{1-\beta}$, the total expected number of steps they take.

An analysis of this algorithm can be found in "Monte Carlo methods in PageRank computation: When one iteration is sufficient."[1] This algorithm provides an unbiased estimator of PageRank with a low variance by removing the randomness of number of walks per node. The MC algorithm uses the whole random walk to do the estimation, hence providing a good estimate of PageRank even with very small values of R.

(c) [4pts]

Assuming that taking each step in a random walk takes a unit amount of time, show that the expected running time of the MC algorithm is $O(\frac{nR}{1-\beta})$.

[1] Avrachenkov, K., Litvak, N., Nemirovsky, D., & Osipova, N. (2007). Monte Carlo methods in PageRank computation: When one iteration is sufficient. SIAM Journal on Numerical Analysis, 45(2), 890-904. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.136&rep=rep1&type=pdf

Comparing the running times in parts (d) and (b), one can see that if the graph is not very sparse, the MC algorithm has a better running time than the Power Iteration.

(d) [15pts]

A small randomly generated graph data (assume graph has no dead-ends) is 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 the graph's strong connectivity. 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. In the following, assume the teleport probability is $1 - \beta = 0.2$.

- Using 40 iterations of Power Iteration, compute the PageRank vector r. What is the running time (CPU time)? Note that the computed vector is still an approximate PageRank vector, but since we are doing a relatively large number of iterations, we will consider it as the "true" PageRank vector, and use it as the baseline in our error computations in the next part.
- Using the MC algorithm, with $R = 1, 3, 5$ compute approximate PageRank vectors $\widetilde{r}$. What is the running time for each value of R? Also, for each value of R, what is the average absolute error for the 10 nodes having the top 10 PageRanks? How about top 30, top 50, and all the nodes? To decrease the noise in the estimates, repeat this 100 times and average the results. If $\widetilde{r}_i^j$ is the Monte Carlo estimate for node i on the jth trial, average absolute error can be defined as

$$\frac{1}{100}\sum_{j=1}^{100}\frac{1}{K}\sum_{i=1}^{K}|\widetilde{r}_i^j - r_i|$$

  Where K represents the number of nodes being compared (i.e. $K = 10$ when comparing top 10 nodes in the pagerank vector).

What to submit

(a) Proof that $||r - r^{(k)}||_1 \leq 2\beta^k$

(b) Show that maximum L_1 error is less than a constant δ when the running time of the Power Iteration method is $O(\frac{m}{\log(1/\beta)})$

(c) Show that expected running time of MC is $O(\frac{nR}{1-\beta})$

(d) This coding part includes submitting the following

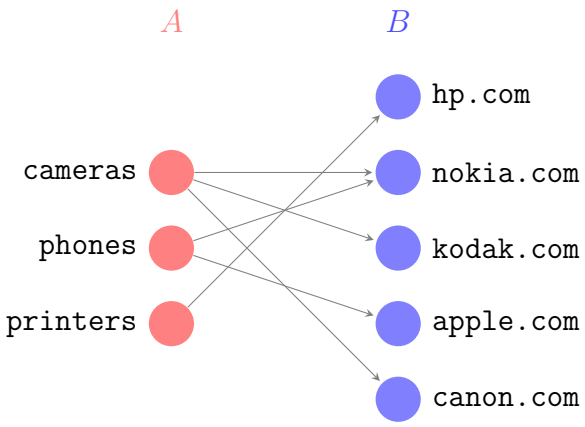

Figure 1: Bipartite graph G of queries A and websites B.

- Submit online to snap and scoryst all the code that you wrote for this question
- Running time of computing the PageRank vector using 40 iterations of Power Iteration
- For MC algorithm, do the following:
  - Compute approximate PageRank vectors with $R = 1, 3, 5$ and list corresponding running times
  - For each value of R, show average absolute error over 100 trials for 10 nodes having top 10 PageRanks, and repeat for top 30, top 50, and all nodes.

3 Similarity Ranking (15 points) [Clifford, Negar]

In this problem, we will study an application of **link analysis** for finding **similarity between items in a bipartite graph**. Note that it is possible to verify your solution by solving this question with Matlab.

Bipartite graph. The bipartite graph G is composed of two sets of vertices A and B with edges between them. One example of such a graph is represented in Figure 1: the set A represents queries, the set B represents URLs, and there is an edge from $a \in A$ to $b \in B$ if the query a led a user to click on the URL b.

Intuition and goal of the problem. The notion we want to capture is that *two items are similar if they are referenced by similar items*. For instance, `cameras` and `phone` are similar because `nokia.com` is clicked on for both of them. Also, `nokia.com` and `apple.com` are similar because they are both clicked on for the search query `phone`. (For now, we are not taking into account the number of clicks, all the edges have a weight of 1.)

Queries and websites can be replaced by other domains. For instance A could be users and B could be items, with edges indicating whether a user bought an item. In this case, the similarity notion—two users are similar if they buy similar items, and two items are similar if they share similar users—is reminiscent of user-user and item-item collaborative filtering. In this exercise, we try to **make use the entire link structure to derive similarity scores**, much like PageRank.

Similarity definitions. We define similarity with a simple *recursive* formulation. Similarity of items from the set A will be denoted s_A, whereas similarity for items from the set B will be denoted s_B. For convenience, let the items from the set A be represented by uppercase letters, and the items from the set B be represented by lowercase letters.

Initialization: The typical strategy to find similarities is to start off with:

$$\begin{cases} s_A(X,X) = 1 & \text{for all} \quad X \in A \\ s_B(x,x) = 1 & \text{for all} \quad x \in B, \end{cases}$$

and:

$$\begin{cases} s_A(X,Y) = 0 & \text{for all} \quad X,Y \in A \quad \text{s.t.} \quad X \neq Y \\ s_B(x,y) = 0 & \text{for all} \quad x,y \in B \quad \text{s.t.} \quad x \neq y. \end{cases}$$

Induction: Then, we define the similarity s_A of any two queries $X, Y \in A$ (such that $X \neq Y$) by:

$$s_A(X,Y) = \frac{C_1}{|O(X)||O(Y)|} \sum_{i=1}^{|O(X)|} \sum_{j=1}^{|O(Y)|} s_B\left(O_i(X), O_j(Y)\right), \tag{3}$$

where $O(X)$ is the set of edges originating from X and $O_i(X) \in B$ is the destination vertex of the i^{th} edge originating from X. The same definitions stand for Y. Finally, C_1 is a "decay" constant < 1.

Similarly, we define the similarity s_B of any two URLs $x, y \in B$ (such that $x \neq y$) by:

$$s_B(x,y) = \frac{C_2}{|I(x)||I(y)|} \sum_{i=1}^{|I(x)|} \sum_{j=1}^{|I(y)|} s_A\left(I_i(x), I_j(y)\right), \tag{4}$$

where $I(x)$ is the set of edges going into x and $I_i(x) \in A$ is the source vertex of the i^{th} edge that points to x. The same definitions stand for y. C_2 is another "decay" constant < 1.

Note that those similarity functions are symmetric, so we only need one score for each pair of items. Thus, the definition of s_A corresponds to $\binom{|A|}{2}$ equations while the definition of s_B corresponds to $\binom{|B|}{2}$ equations.

Similarity calculus. We iterate by executing equations (3) and (4) for all pairs as many times as required until the values of s_A and s_B converge. Note that for each iteration, we are using the similarity values from the previous iteration. You can view the $\binom{|A|}{2} + \binom{|B|}{2}$ equations as one batch of assignments. So, if we do k iterations, we execute this batch k times.

It can be shown that this procedure converges to the right answer.

(a) [5pts]

Let $C_1 = C_2 = 0.8$. Execute the batch of equations 3 times for the graph shown in Figure 1 by hand (or with a simple program).

Find the values of $s_A(\texttt{camera}, \texttt{phone})$ and $s_A(\texttt{camera}, \texttt{printer})$ after 3 iterations.

Your writeup should contain the detailed steps for at least one iteration of the calculus for one pair of items, as well as all the similarity intermediate results (*i.e.* after step 1, step 2 and step 3).

(b) [5pts]

The similarity scores above assume that all edges are equally relevant. Let's say we also have information about how many times a URL is clicked for a given query (or equivalently, many users bought how many items). For a pair (X, y), we denote this information by the weight $W_{(X,y)}$.

Modify the two equations (3) and (4) to take into account this information for similarity.

(c) [5pts]

This similarity computation scheme has its problems.

Let's say $K_{2,1}$ is the complete bipartite graph[2] with 2 nodes in A and 1 node in B. Similarly $K_{2,2}$ is the complete bipartite graph with 2 nodes in A and 2 nodes in B. Consider the two nodes in A. Intuitively, it seems that these two nodes should have a higher score in $K_{2,2}$ since there is double the evidence that they are similar. However, this is not the case.

Compute the similarity scores using the approach from part (a) for the two nodes in A after 3 iterations for both graphs. How do they compare?

Do the calculus by hand (or with a simple program) for $C_1 = C_2 = 0.8$. Your write up should contain the detailed steps for at least one iteration, as well as all the similarity intermediate

[2] A *complete* bipartite graph is a special kind of bipartite graph where every vertex of the first node set A is connected to every vertex of the second nodes set B.

results.

(Note: It can be shown that this relationship holds for all iterations.)

What to submit

(a) The values of $s_A(\texttt{camera}, \texttt{phone})$ and $s_A(\texttt{camera}, \texttt{printer})$ after 3 iterations, with the detailed calculus for at least one iteration, as well as all the intermediate similarity scores

(b) An updated version of equations (3) and (4) that take into account the weight W of the edges

(c) For $K_{2,1}$ and $K_{2,2}$, the values of s_A for the pair of nodes in A after 3 iterations, with the detailed calculus for at least one iteration, as well as all the intermediate similarity scores

4 Dense Communities in Networks (25 points) [Janice, Dima, Nat]

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 the nodes with low degrees do not contribute much to the density of a dense subgraph, hence they can be removed without significantly influencing the density.

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

(a) [5 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) [5 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 $\rho(\tilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^*(G)$.

(c) [15 points]

An undirected graph specified as a list of edges, is provided at http://snap.stanford.edu/class/cs246-data/livejournal-undirected.txt.zip.

The data set consists of $499,923$ vertices and $7,794,290$ edges. Treat each line in the file as an undirected edge connecting the two given vertices.

Implement the algorithm (in any programming language) described above using a streaming model. Here, you cannot load the graph into memory but you can read through the file as many times as is needed (essentially this means that you are allowed to use $\mathcal{O}(n)$ space but not $\mathcal{O}(m)$). For this problem, let the density of an empty set be 0. Note: You do not need to read line by line. You may choose to read chunks of the file in at once for efficiency. **Include your code on Scoryst and submit it to Snap.**

Now answer the following questions:

i. The number of iterations needed to find a dense subgraph depends upon the value of ϵ. Plot a graph showing how many iterations it takes to calculate the first dense subgraph when $\epsilon = \{0.1, 0.5, 1, 2\}$ and compare it with the theoretical bounds shown in (a).

ii. When $\epsilon = 0.05$, plot separate graphs showing $\rho(S_i), |E(S_i)|$ and $|S_i|$ as a function of i where i is the iteration of the while loop.

iii. The algorithm above only describes how to find one dense component (or community). It is also possible to find multiple components by running the above algorithm to find the first dense component and then deleting all vertices (and edges) belonging to that component. To find the next dense component, run the same algorithm on the modified graph. Plot separate graphs showing $\rho(\tilde{S}_j), |E[\tilde{S}_j]|$ and $|\tilde{S}_j|$ as a function of j where j is the current community that has been found. You can stop if 20 communities have already been found. Have $\epsilon = 0.05$.

Note: Expected runtime will be in the order of minutes though inefficient code (especially Python) may result in longer computation time. Efficient code is good but not required. Note also that the simulation of part c.iii can take over an hour because you have to run the algorithm to find communities 20 times.

To test out your code, a smaller dataset consisting of 50 vertices and 90 edges is provided at

http://snap.stanford.edu/class/cs246-data/livejournal-undirected-small.txt.zip.

What to submit

Include the code in your Scoryst pdf submission and also upload code to SNAP.

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

(c) i.
- Plot of number of iterations needed to find a dense subgraph. One line for theoretical and one line for actual.
- Comparison of plot with theoretical bounds

ii. 3 Separate Graphs:
- Graph showing $\rho(S_i)$ as a function of i
- Graph showing $|E(S_i)|$ as a function of i
- Graph showing $|S_i|$ as a function of i

iii. 3 Separate Graphs:
- Graph showing $\rho(\tilde{S}_j)$ as a function of j
- Graph showing $|E(\tilde{S}_j)|$ as a function of j
- Graph showing $|\tilde{S}_j|$ as a function of j