CS246: Mining Massive Data Sets **Winter 2017**

Problem Set 3

Due 11:59pm February 23, 2017

Only one late period is allowed for this homework (11:59pm 2/28).

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/hw3_template.tex
http://web.stanford.edu/class/cs246/homeworks/hw3/hw3_template.pdf

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 (35 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 number 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) [10 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?

(b) [25 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 $k = 20$, $\lambda = 0.1$ 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.train.txt discussed below should be less than 65000 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 Understanding the Effect of Merging Strategies on Clustering (10 points)

We learned in class that different merging strategies result in different clustering outcomes. In this problem, we explore how various merging strategies impact the clustering outcomes by means of an example. Consider a small 1-dimensional dataset comprising 9 data points: $\mathcal{P} = \{p_1, p_2, \cdots, p_9\} = \{0.68, 0.72, 0.79, 0.87, 0.92, 0.99, 1.01, 1.22, 1.45\}$. This dataset captures the smoking habits of 9 different individuals. Each of the numbers represent a quantity called the *Normalized Smoking Rate (NSR)* which is defined as the ratio of the mean number of times an individual i smokes per day to the mean number of times an average individual in the U.S. smokes. This implies that if $p_i > 1.0$ for some individual i, then i smokes more than an average person in the U.S. In our dataset, there are three such individuals who smoke more than an average person the U.S. A survey done by the health department indicated that any individual with an NSR score > 1.0 is at a high risk of lung cancer. So, our dataset captures two sub-populations: people with low risk of lung cancer (p_1 to p_6) and people with high risk of lung cancer (p_7 to p_9).

We were so enthusiastic about seeing the results of the clustering on this dataset that we already ran an unknown clustering algorithm on this data. The resulting clusters are shown in Figure 1. As can be seen, there are three clusters each of which comprises of three datapoints. Our goal is to group all the datapoints into two clusters. In order to achieve that, you should now decide which pair of the three existing clusters in Figure 1 should be merged so that we are left with two clusters of the data.

Consider the following merging strategies to decide which pair of the existing clusters you would like to merge:

- Centroid based merging: Merge the pair of clusters with the smallest distance between their centroids.
- Closest member based merging: Take the distance between two clusters to be the minimum of the distances between any two points, one chosen from each cluster. Merge the pair of clusters with minimum distance.

Note: Centroid of a cluster is computed as the average of the all the datapoints in that cluster (rounded to two decimal places). Through out this question, whenever we use the word distance, we are referring to the Euclidean distance metric.

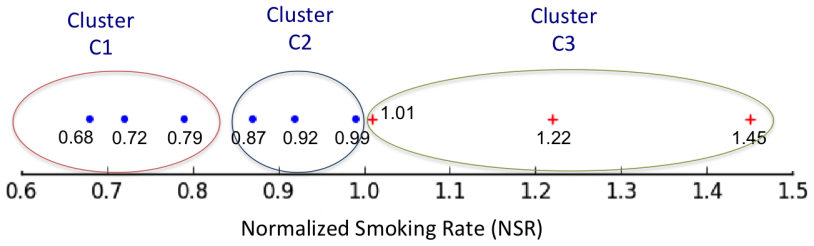

Figure 1: Clusters already obtained from an unknown clustering algorithm

(a) [4 points]

Which pair of clusters shown in Figure 1 will be merged by each of the strategies outlined above (Centroid based merging, Closest member based merging)? Which merge strategy will leave behind two clusters such that one of the clusters corresponds to low risk of lung cancer and the other represents high risk of lung cancer?

(b) [4 points]

A merge strategy is called *stable* w.r.t some point p_i and some existing set of clusters S_c if assuming that p_i is absent from the dataset (as well as from its currently assigned cluster) does not change the outcome of the strategy. Of all the merge strategies outlined above, which ones are *stable* w.r.t the point $p_6 = 0.99$ and the existing set of clusters shown in Figure 1 i.e which of the merge strategies are unaffected if we assume that p_6 does not exist in the data and also assume that cluster $C2$ (in Figure 1) just has two data points p_4 and p_5.
Note: When applying centroid based merging, centroid of cluster $C2$ should be computed by assuming that p_6 does not exist.

(c) [2 points]

If we give you a new NSR dataset which is noisy i.e the NSR scores have been miscalculated for about 5% of individuals in this new dataset, would you prefer using a stable merge strategy instead of an unstable one? Explain your reasons.

What to submit:

(i) Pair of clusters merged for each of the strategies [for 3(a)]

(ii) Indicate the strategy that creates two groups: low risk and high risk of lung cancer [for 3(a)]

(iii) Indicate which strategy (or strategies) are stable [for 3(b)]

(iv) State which strategy would you prefer and why [for 3(c)]

3 Clique-Based Communities (25 points)

Imagine an undirected graph G with nodes $2, 3, 4, \ldots, 1000000$. (Note that there is no node 1.) There is an edge between nodes i and j if and only if i and j have a common factor other than 1. Put another way, the only edges that are missing are those between nodes that are relatively prime; e.g., there is no edge between 15 and 56.

We want to find communities by starting with a clique (not a bi-clique) and growing it by adding nodes. However, when we grow a clique, we want to keep the density of edges at 1; i.e., the set of nodes remains a clique at all times. A *maximal clique* is a clique for which it is impossible to add a node and still retain the property of being a clique; i.e., a clique C is maximal if every node not in C is missing an edge to at least one member of C.

(a) [5 points]

Prove that if i is any integer greater than 1, then the set C_i of nodes of G that are divisible by i is a clique.

(b) [10 points]

Under what circumstances is C_i a maximal clique? Prove that your conditions are both necessary and sufficient. (Trivial conditions, like "C_i is a maximal clique if and only if C_i is a maximal clique," will receive no credit.)

(c) [10 points]

C_6 is the set of nodes $6, 12, 18, \ldots, 999996$. C_6 is not maximal, and we can add nodes to it and still keep the density of edges at 1, until we arrive at a maximal clique. This process is not deterministic, since we usually have many choices of the next node to add. However, it turns out that there are only two maximal cliques that we can reach when no more nodes can be added. What are these two cliques? What determines which of the two cliques we reach? Prove your statements.

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