CS246: Mining Massive Datasets

Assignment number: 3

Submission time: ______ **and date:** ______

Fill in and include this cover sheet with each of your assignments. It is an honor code violation to write down the wrong time. Assignments are due at 9:30 am, either handed in at the beginning of class or left in the submission box on the 1^{st} floor of the Gates building, near the east entrance. Failure to include the coversheet with you assignment will be penalized by 2 points.

Each student will have a total of *two* free late periods. *One late period expires at the start of each class.* (Assignments are due on Thursdays, which means the first late period expires on the following Tuesday at 9:30am.) Once these late periods are exhausted, any assignments turned in late will be penalized 50% per late period. However, no assignment will be accepted more than *one* late period after its due date. (If an assignment is due to Thursday then we will not accept it after the following Thursday.)

Your name: Blaž Sovdat
Email: blaz.sovdat@gmail.com **SUNet ID:** ______

Collaborators: ______

I acknowledge and accept the Honor Code.

(Signed) ______

(For CS246 staff only)

Late days: 0 1

Section	Score
1	
2	
3	
4	
Total	

Comments: /

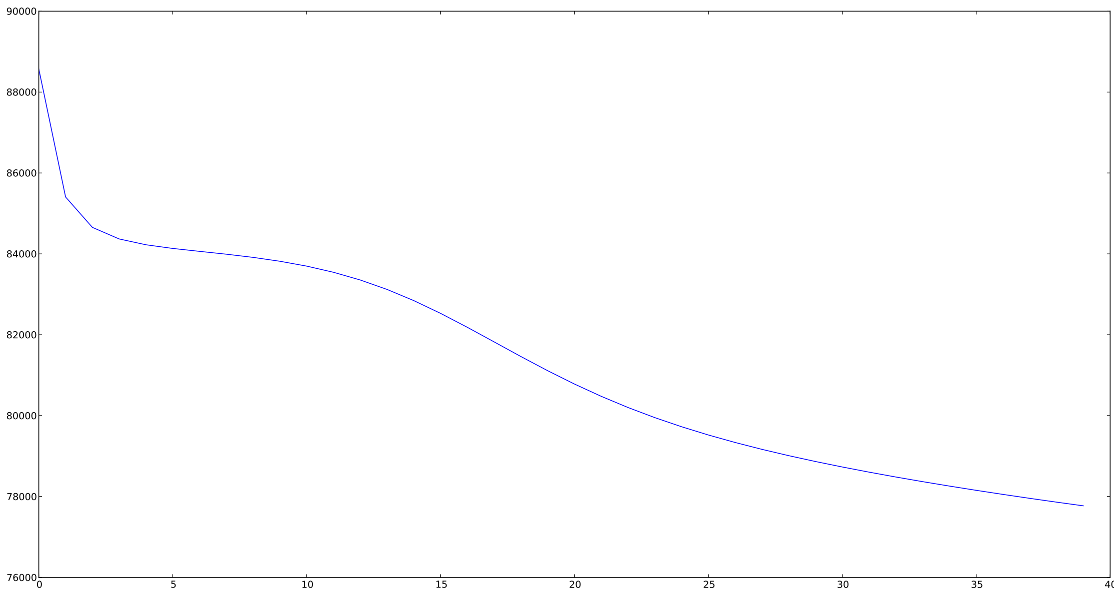

Figure 1: Error in each of the 40 iterations of our algorithm.

Note. You can find all source code in appendix A at the end of the document.

1 Latent Features for Recommendations

(a) We have $\epsilon_{iu} = \frac{\partial}{\partial R_{iu}} E = 2(R_{iu} - q_i p_u^{\mathrm{T}})$. This means ϵ_{iu} is the prediction error (times two) we make for prediction of rating user u made for item i. (Note that in our code we use $\epsilon_{iu} = R_{iu} - q_i p_u^{\mathrm{T}}$.) The update equations are as follows:

$$q_i := q_i + \mu_1(\epsilon_{iu} p_u - \lambda_1 q_i),$$
$$p_u := p_u + \mu_2(\epsilon_{iu} q_i - \lambda_2 p_u).$$

In our case we have $\mu_1 = \mu_2 = \eta$ and $\lambda_1 = \lambda_2 = \lambda$.

(b) See listing 1 for our implementation of the stohastic gradient descent algorithm. See figure 1 for plot of the value of the objective function on the training set as a function of the number of iterations. We initialized P and Q with values $[0, \sqrt{5/k}]$, picked uniformly at random. Furthermore, we found that setting learning rate $\eta := 0.01565$ works well.

(c) Define the training error as $E_{\mathrm{tr}} := \sum_{(i,u) \in \mathrm{train}} (R_i u - q_i p_u^{\mathrm{T}})^2$ and the test error as $E_{\mathrm{te}} := \sum_{(i,u) \in \mathrm{test}} (R_i u - q_i p_u^{\mathrm{T}})^2$. See listing 2 for the modified code. See figure 2 for errors with and without regularization on the test set and figure 3 for errors with and without regularization on the training set. True statements are the following ones:

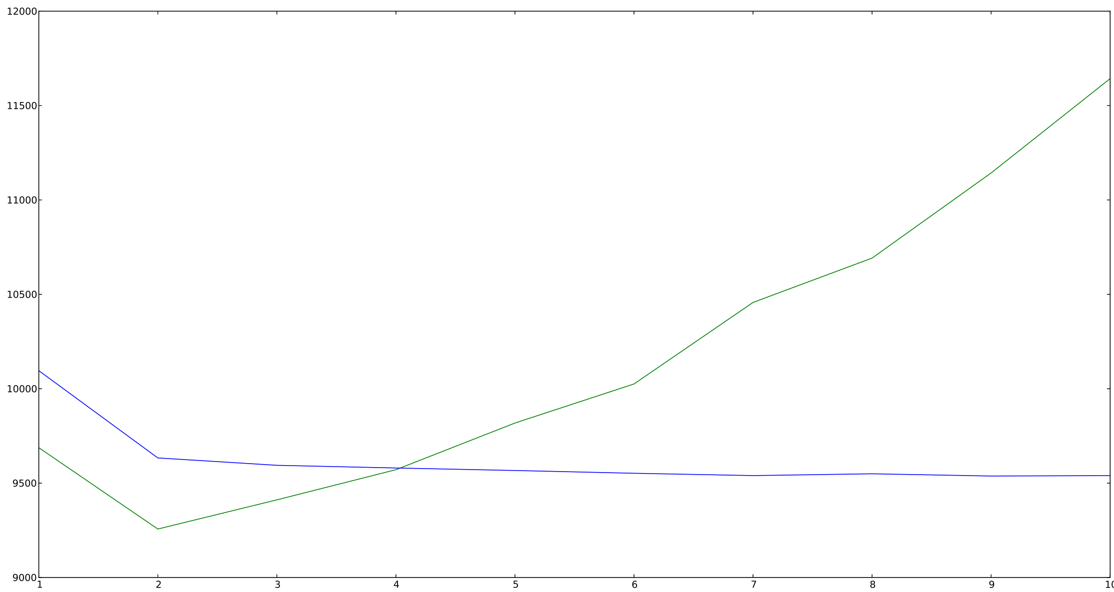

Figure 2: Plot for E_{te} with no regularization (green) and E_{te} with $\lambda = 0.2$ (blue) as a function of $k \in \{1, 2, \ldots, 10\}$.

- B: Regularization decreases the test error for $k \geq 5$.
- D: Regularization increases the training error for all (or almost all) k.
- H: Regularization decreases overfitting.

2 PageRank Computation

(a) Define $r^{(0)} = \frac{1}{n}e$ and $r^{(k)} = \frac{1-\beta}{n}e + \beta M r^{(k-1)}$ for $k > 0$ and $\beta \in (0, 1)$, where e is the unit vector of dimension n. Furthermore let r be the PageRank vector, so we have $r = \frac{1-\beta}{n}e + \beta M r$. We now prove $\|r - r^{(k)}\|_1 \leq 2\beta^k$ for all $k \geq 0$ by induction. Fist note $\|r - r^{(0)}\|_1 \leq 2$, so we have the base case. (Norm is largest when $p_i = 1$ for some i and $p_j = 0$ for $i \neq j$. So $\|r - r^{(0)}\|_1 \leq 1 - 1/n + n \cdot 1/n = 2(n-1)/n \leq 2$.) Now suppose

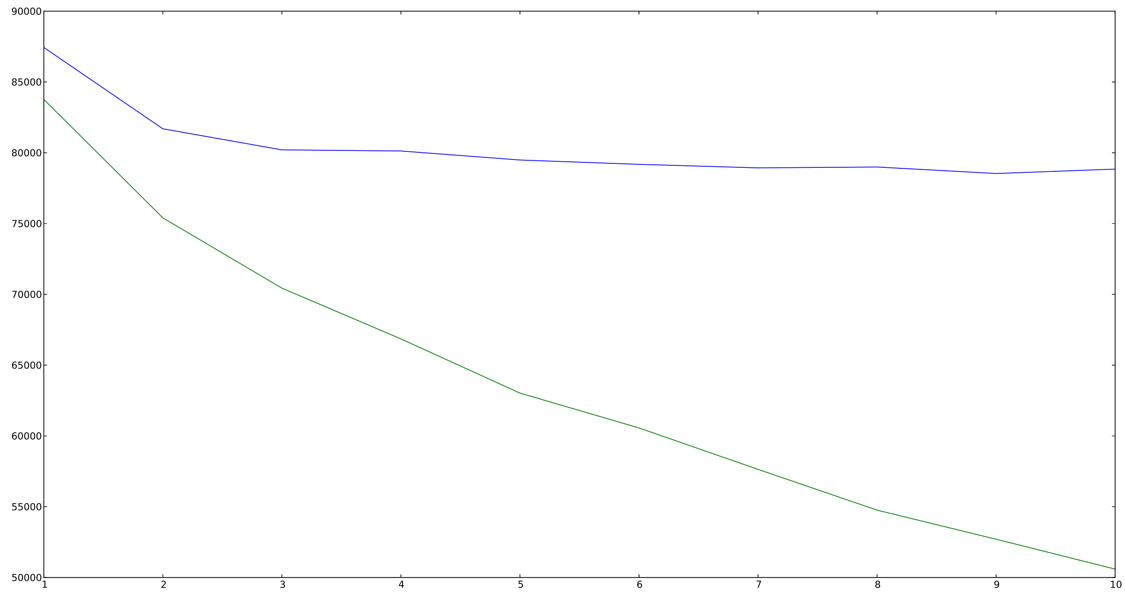

Figure 3: Plot for E_{tr} with no regularization (green) and E_{tr} with $\lambda = 0.2$ (blue) as a function of $k \in \{1, 2, \ldots, 10\}$.

$\|r - r^{(\ell)}\|_1 \leq 2\beta^\ell$ for $\ell = 0, 1, \ldots, k$. We now do the inductive step:

$$\begin{aligned}
\|r - r^{(k+1)}\|_1 &= \|\frac{1-\beta}{n}e + \beta M r - \frac{1-\beta}{n}e - \beta M r^{(k)}\|_1 \\
&= \|\beta M(r - r^{(k)})\|_1 \\
&= \beta\|M(r - r^{(k)})\|_1 \\
&\leq \beta\|M\|_1\|r - r^{(k)}\|_1 \\
&\leq 2\beta^{k+1},
\end{aligned}$$

establishing the inequality. (Note that for the matrix L_1 norm we have $\|M\|_1 = \max_{1\leq j\leq n}\sum_{i=1}^n |M_{ij}| \leq 1$, because M is column stohastic. We also used $\|Mv\|_1 \leq \|M\|_1\|v\|_1$ for matrix M and vector v.)

(b) Suppose $\|r - r^{(k)}\|_1 \leq 2\beta^k$ and let $\delta \in (0, 1)$ be constant. We then solve

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

for k and get $k > \log(\delta/2)/\log(1/\beta)$. Since δ is constant this means we need $k = O(1/\log(1/\beta))$ iterations to get error at most δ. Because the cost of each iteration is $O(m)$, we have that the total running time required to get the desired error is $O(m/\log(1/\beta))$. (We can compute $(1-\beta)e/n + \beta M r^{(k)}$ in $O(m)$.) This means that after $O(m/\log(1/\beta))$ steps we have error less than δ.

(c) Show $\mathbb{E}[\widetilde{r}_j] = \frac{1-\beta}{nR}\,\mathbb{E}[\text{visits}(j)] = r_j$, where r_j is j-th component of the true PageRank vector and $\mathbb{E}[\widetilde{r}_j]$ is the expected number of times the node j is visited, taken over all nR random walks. Recall that for PageRank we have

$$r_j = \frac{1-\beta}{n} + \beta \sum_{(i\to j)\in E} \frac{r_i}{\deg(i)},$$

which is equivalent to $r = \frac{1-\beta}{n}e + \beta M r$. Could we somehow get $\mathbb{E}[\widetilde{r}] = \frac{1-\beta}{n}e + \beta M\,\mathbb{E}[\widetilde{r}]$?

(d) We now show that the running time of the MC algorithm is $O(\frac{nR}{1-\beta})$ if we run it R times from each node on a graph with n vertices. First, let $T(v)$ be running time of the algorithm when run from the vertex v. After updating the count for v, it will continue the walk with probability β and terminate with probability $1-\beta$, so we clearly have $T(v) = O(1) + \beta T(v)$, which gives us the geometric series

$$T(v) = \sum_{k\geq 0} \beta^k O(1) = O(\frac{1}{1-\beta}).$$

Since we run the algorithm R times for each node, we have the total running time of $T(v)nR = O(\frac{nR}{1-\beta})$.

(e) We implemented both PageRank with Power iteration and the MC algorithm in python; for the code see listing 3.

- The running time is 0.008119 seconds. See listing 3.
- For code for this question see listing 3. To compute the running times we took the mean running time of 50 runs for each $R = 1, 3, 5$. For $R = 1$ the running time is 0.0031599 seconds. For $R = 3$ the running time is 0.005479 seconds. For $R = 5$ the running time is 0.00712 seconds. The absolute errors — see listing 3 for implementation — were as in table 1, where error is defined as

  $$E_K := \frac{1}{K}\sum_{i=1}^{K} |\widetilde{r}_i - r_i|,$$

  for approximate PageRank vector $\widetilde{r}$ and "true" PageRank r. (See HW text for detailed description of the error metric.)

3 Similarity Ranking

(a) Let $C_1 := C_2 := 0.8$. We compute 3 steps of similarity for $s_A(\texttt{camera}, \texttt{phone})$ and $s_A(\texttt{camera}, \texttt{printer})$.

R	K	Error
1	10	0.0106990249484
1	30	0.00947132554345
1	50	0.00766502952898
1	All	0.0073999233725
3	10	0.00989902494841
3	30	0.00898243665457
3	50	0.00694048754126
3	All	0.00695841712359
5	10	0.00983703989898
5	30	0.00843815848515
5	50	0.00682750626194
5	All	0.00641664075857

Table 1: Average absolute errors.

Step 1. We have

$$\begin{aligned}
s_A(\texttt{camera},\texttt{phone}) &= \frac{0.8}{6}\left(s_B(\texttt{nokia},\texttt{nokia}) + s_B(\texttt{nokia},\texttt{apple}) + s_B(\texttt{kodak},\texttt{nokia}) + s_B(\texttt{kodak},\texttt{apple}) + s_B(\texttt{canon},\texttt{nokia}) + s_B(\texttt{canon},\texttt{apple})\right)\\
&= \frac{0.8}{6}\cdot 1 = \frac{4}{5\cdot 6} = 2/15\\
s_A(\texttt{camera},\texttt{printer}) &= \frac{0.8}{3}\left(s_B(\texttt{nokia},\texttt{hp}) + s_B(\texttt{kodak},\texttt{hp}) + s_B(\texttt{canon},\texttt{hp})\right) = 0\\
s_A(\texttt{phone},\texttt{printer}) &= \frac{0.8}{2}\left(s_B(\texttt{nokia},\texttt{hp}) + s_B(\texttt{apple},\texttt{hp})\right) = 0\\
s_B(\texttt{nokia},\texttt{apple}) &= \frac{0.8}{2}\left(s_A(\texttt{phone},\texttt{phone}) + s_A(\texttt{phone},\texttt{camera})\right) = 2/5\\
s_B(\texttt{nokia},\texttt{hp}) &= \frac{0.8}{2}\left(s_A(\texttt{phone},\texttt{printer}) + s_A(\texttt{camera},\texttt{printer})\right) = 0\\
s_B(\texttt{nokia},\texttt{kodak}) &= \frac{0.8}{2}\left(s_A(\texttt{phone},\texttt{camera}) + s_A(\texttt{camera},\texttt{camera})\right) = 2/5\\
s_B(\texttt{nokia},\texttt{canon}) &= \frac{0.8}{2}\left(s_A(\texttt{phone},\texttt{camera}) + s_A(\texttt{camera},\texttt{camera})\right) = 2/5\\
s_B(\texttt{kodak},\texttt{apple}) &= \frac{0.8}{1}\left(s_A(\texttt{phone},\texttt{camera})\right) = 0\\
s_B(\texttt{kodak},\texttt{hp}) &= \frac{0.8}{1}\left(s_A(\texttt{printer},\texttt{camera})\right) = 0\\
s_B(\texttt{kodak},\texttt{cannon}) &= \frac{0.8}{1}\left(s_A(\texttt{camera},\texttt{camera})\right) = 4/5\\
s_B(\texttt{canon},\texttt{apple}) &= \frac{0.8}{1}\left(s_A(\texttt{phone},\texttt{camera})\right) = 0\\
s_B(\texttt{canon},\texttt{hp}) &= \frac{0.8}{1}\left(s_A(\texttt{printer},\texttt{camera})\right) = 0\\
s_B(\texttt{apple},\texttt{hp}) &= \frac{0.8}{1}\left(s_A(\texttt{phone},\texttt{printer})\right) = 0
\end{aligned}$$

Step 2. We now use similarities from step 1.

$$s_A(\texttt{camera},\texttt{phone}) = \frac{0.8}{6}\left(1+3\frac{2}{5}\right) = 22/75 \qquad s_A(\texttt{camera},\texttt{printer}) = 0$$

$$s_A(\texttt{phone},\texttt{printer}) = 0 \qquad s_B(\texttt{nokia},\texttt{apple}) = 34/75$$

$$s_B(\texttt{nokia},\texttt{hp}) = 0 \qquad s_B(\texttt{nokia},\texttt{kodak}) = 34/75$$

$$s_B(\texttt{nokia},\texttt{canon}) = 34/75 \qquad s_B(\texttt{kodak},\texttt{apple}) = 8/75$$

$$s_B(\texttt{kodak},\texttt{hp}) = 0 \qquad s_B(\texttt{kodak},\texttt{cannon}) = 4/5$$

$$s_B(\texttt{canon},\texttt{apple}) = 8/75 \qquad s_B(\texttt{canon},\texttt{hp}) = 0$$

$$s_B(\texttt{apple},\texttt{hp}) = 0$$

Step 3. Finally we compute 3 step using similarities from the previous one:

$$s_A(\texttt{camera},\texttt{phone}) = \frac{4}{5\cdot 6}(1+\frac{2\cdot 8}{75}+\frac{3\cdot 34}{75}) = 0.343 \quad s_A(\texttt{camera},\texttt{printer}) = 0$$

(b) 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 $W_{(X,y)}$. We propose the following:

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

$$s_B(x,y) = \frac{C_2}{\sum_{i=1}^{|I(x)|}\sum_{j=1}^{|I(y)|} W_{x,I_i(x)}W_{y,I_i(y)}} \sum_{i=1}^{|I(x)|}\sum_{j=1}^{|I(y)|} W_{x,I_i(x)}W_{y,I_i(y)}s_B(I_i(X),I_j(y)).$$

(c) Let $C_1 := C_2 := 0.8$. We compute 3 iterations of $s_A(a_1,a_2)$ for complete bipartite graphs $K_{2,1}$ and $K_{2,2}$. For $K_{2,1}$ with $A=\{a_1,a_2\}$ and $B=\{b_1\}$ we have:

Step 1) Note $s_B(b_1,b_1)=1$ by "definition". (We would not need to even write these down as they never change; but the author felt including them would make no harm.)

$$s_A(a_1,a_2) = 0.8s_B(b_1,b_1) = 0.8 = 4/5 \qquad s_B(b_1,b_1) = 1.$$

Step 2) We now use previous scores:

$$s_A(a_1,a_2) = 0.8s_B(b_1,b_1) = 4/5 \qquad s_B(b_1,b_1) = 1.$$

Step 3) Similarly for the final step

$$s_A(a_1,a_2) = 0.8s_B(b_1,b_1) = 4/5 \qquad s_B(b_1,b_1) = 1.$$

We now repeat the procedure for $K_{2,2}$ with $A = \{a_1, a_2\}$ and $B = \{b_1, b_2\}$:

Step 1) Using initialization values we have

$$s_A(a_1, a_2) = \frac{0.8}{4}(s_B(b_1, b_1) + s_B(b_1, b_2) + s_B(b_2, b_2) = \frac{0.8}{4}2 = 4/10,$$
$$s_B(b_1, b_2) = \frac{0.8}{4}(s_A(a_1, a_1) + s_A(a_1, a_2) + s_A(a_2, a_2) = \frac{0.8}{4}2 = 4/10.$$

Step 2) We now use scores from step 1:

$$\begin{aligned} s_A(a_1, a_2) &= \frac{0.8}{4}(s_B(b_1, b_1) + s_B(b_1, b_2) + s_B(b_2, b_2) = \frac{0.8}{4}(2 + 4/10) \\ &= 24/50, \\ s_B(b_1, b_2) &= \frac{0.8}{4}(s_A(a_1, a_1) + s_A(a_1, a_2) + s_A(a_2, a_2) = \frac{0.8}{4}(2 + 4/10) \\ &= 24/50. \end{aligned}$$

Step 3) Finally, compute third step using scores from step 2:

$$\begin{aligned} s_A(a_1, a_2) &= \frac{0.8}{4}(s_B(b_1, b_1) + s_B(b_1, b_2) + s_B(b_2, b_1) + s_B(b_2, b_2) = \frac{0.8}{4}(2 + \frac{24}{50}) \\ &= 124/250, \\ s_B(b_1, b_2) &= \frac{0.8}{4}(s_A(a_1, a_1) + s_A(a_1, a_2) + s_A(a_2, a_1) + s_A(a_2, a_2) = \frac{0.8}{4}(2 + \frac{24}{50}) \\ &= 124/250. \end{aligned}$$

For $K_{2,1}$ the scores $s_A(a_1, a_2)$ do not change through iterations, while for $K_{2,2}$ they converge to $1/2$.

4 Dense Communities in Networks

We here use the well-knonw handshaking lemma, stating $2|E(G)| = \sum_{v \in V(G)} \deg_G(v)$. (For example, $\rho(S) = \frac{1}{2|S|} \sum_{v \in S} \deg_S(v)$.)

(a) We first prove lower bounds on the size of $A(S)$ and on the number of iterations of the algorithm.

(i) We show $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$. Note that we have

$$2|E[S]| = \sum_{v \in S} \deg_S(v) \geq \sum_{v \in S \setminus A(S)} \deg_S(v) \geq 2(1+\epsilon)\frac{|E[S]|}{|S|}(|S| - |A(S)|),$$

implying $|S| \geq (|S| - |A(S)|)(1 + \epsilon)$, which immediately yields $|A(S)| \geq \frac{\epsilon}{1+\epsilon}|S|$. (We have $\sum_{v \in S \setminus A(S)} \deg_S(v) \geq 2(1 + \epsilon)\frac{|E[S]|}{|S|}(|S| - |A(S)|)$ because by definition each node in $S \setminus A(S)$ has degree at least $2(1 + \epsilon)\rho(S)$ and because S and $A(S)$ are disjoint there are $|S| - |A(s)|$ nodes.)

(ii) Let $|S|$ be the current size. In next iteration we are left with at most $\frac{\epsilon}{1+\epsilon}|S|$ nodes. This means $|S| - \frac{\epsilon}{1+\epsilon}|S| = \frac{1}{1+\epsilon}|S|$ for each iteration, i.e., in each iteration the size decreases by factor of at least $(1+\epsilon)$. Since $|S| = |V| = n$ initially, this means we will obviously make at most $1+\log_{1+\epsilon} n$ steps before $S = \emptyset$. (If $n = 1$ the algorithm clearly needs one step, while $\log_{1+\epsilon} n = 0$.)

(b) We now show that the density of the set the algorithm returns is at most $2(1+\epsilon)$ times smaller than $\rho^\star(G)$.

(i) Let $S^\star := \underset{S\subseteq V(G)}{\arg\max} \frac{|E[S]|}{|S|}$ and for the sake of contradiction assume there exists $v \in S^\star$ such that $\deg_{S^\star}(v) \leq \rho^\star(G) - 1$. Now note that

$$\begin{aligned}\rho(S^\star\backslash\{v\}) &= \frac{|E[S^\star]| - \deg_{S^\star}(v)}{|S^\star| - 1} \\ &\geq \frac{|E[S^\star]| - \rho^\star(G) + 1}{|S^\star| - 1} \\ &= \frac{\rho^\star(G)(|S^\star| - 1) + 1}{|S^\star| - 1} \\ &= \rho^\star(G) + \frac{1}{|S^\star| - 1},\end{aligned}$$

meaning that $S^\star\backslash\{v\}$ is denser than $S^\star$. This contradicts the maximality assumption, establishing $\deg_{S^\star}(v) \geq \rho^\star(G)$ for all $v \in S^\star$. We used $|E[S^\star]| = |S^\star|\rho^\star(G)$.

(ii) Consider the first iteration of the while loop so that there is a node $v \in S^\star \cap A(S)$. We show that then $2(1+\epsilon)\rho(S) \geq \rho^\star(G)$. Since $v \in A(S)$ we have $\deg_S(v) \geq 2(1+\epsilon)\rho(S)$. Because this is the first iteration such that $S^\star \cap A(S) \neq \emptyset$, we have $S^\star \subseteq S$ and thus $\deg_S(v) \geq \deg_{S^\star}(v)$. (Because S is superset of $S^\star$, the vertex v can only "touch" more edges in the graph spanned by S than in the graph spanned by $S^\star$, thus $\deg_S(v) \geq \deg_{S^\star}(v)$.) We now trivially have

$$\rho^\star(G) \leq \deg_{S^\star}(v) \leq \deg_S(v) \leq 2(1+\epsilon)\rho(S),$$

establishing the desired inequality.

(iii) We now conclude $\rho(\widetilde{S}) \geq \frac{1}{2(1+\epsilon)}\rho^\star(G)$. This must be the case, because $\widetilde{S}$ is the set S that maximizes $\rho(S)$ and we know from previous "bullet" that there exists (we compute it in one of the iterations of the algorithm) at least one S such that $2(1+\epsilon)\rho(S) \geq \rho^\star(G)$, because $S^\star \subseteq V$ and we start with $S = V$ and keep removing vertices until we are left with $S = \emptyset$.

(c) See listing 4 for the implementation of the algorithm.

(i) See figure 4 for iterations needed (blue) and the theoretical bounds (green). We see that in practice the algorithm performs — in terms of number of iterations as

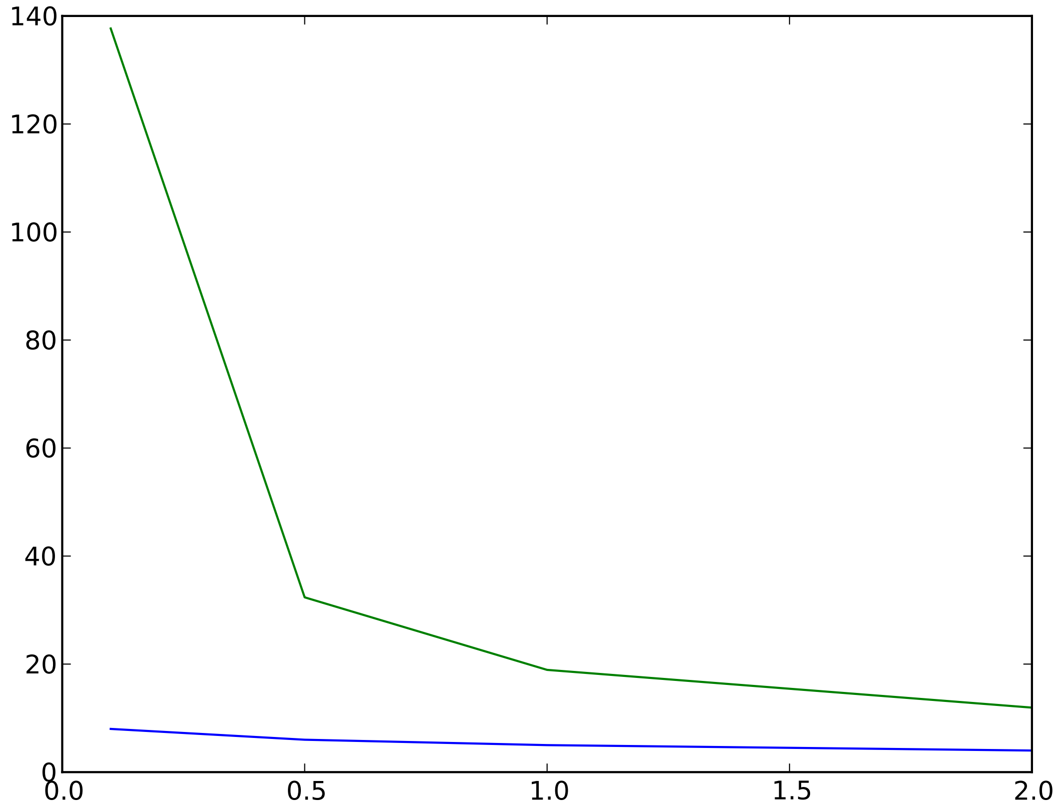

Figure 4: Number of steps (blue) and theoretical bounds (green) for various epsilons (epsilons are on x-axis).

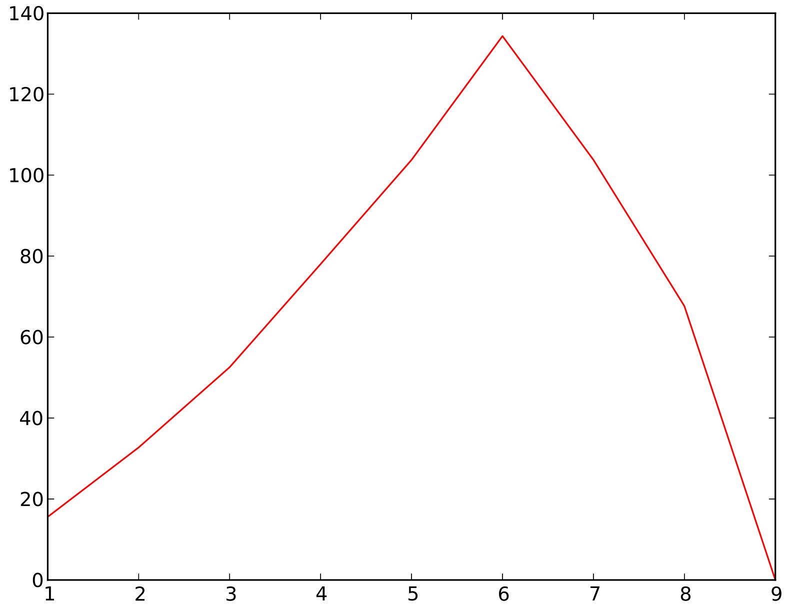

Figure 5: Density $\rho(S_i)$ as a function of the number of steps.

a function of ϵ — much better than what the theoretical upper bound guarantees us. (Note: We plotted $1 + \log_{1+\epsilon}(n)$, not $\log_{1+\epsilon}(n)$.)

(ii) See figure 5 for density, figure 6 for the number of edges, and figure 7 for the number of vertices.

(iii) See figure 8 for density, figure 9 for the number of edges, and figure 10 for the number of vertices.

A Source code

This appendix includes all the source code that we reference throughout the document.

A.1 Latent Features for Recommendations

Listing 1: SGD code for HW3Q1 item b.

```
//
// Compile with 'g++ -std=c++0x -O3 sgd.cpp -o sgd'
// -- Blaz Sovdat, 18 feb 2014
//

#include <algorithm>
```

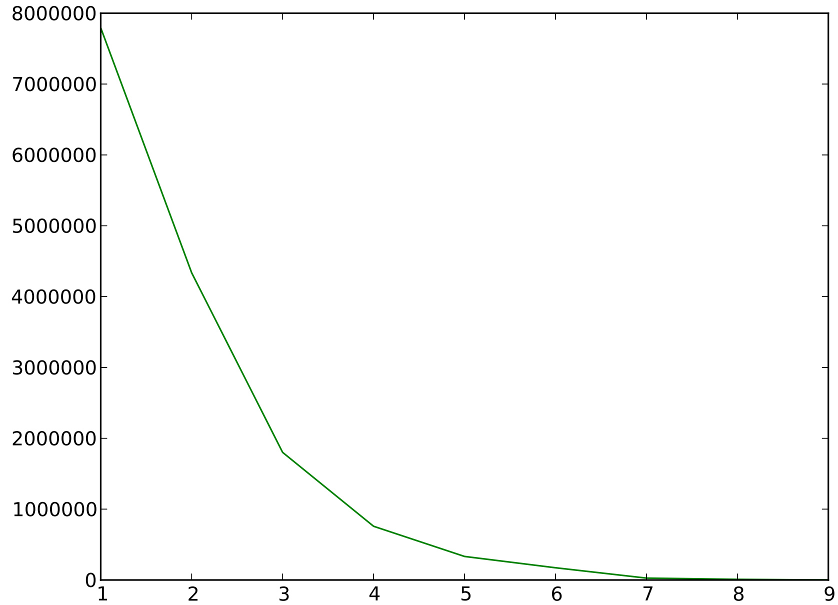

Figure 6: Number of edges $|E[S_i]|$ as a function of the number of steps.

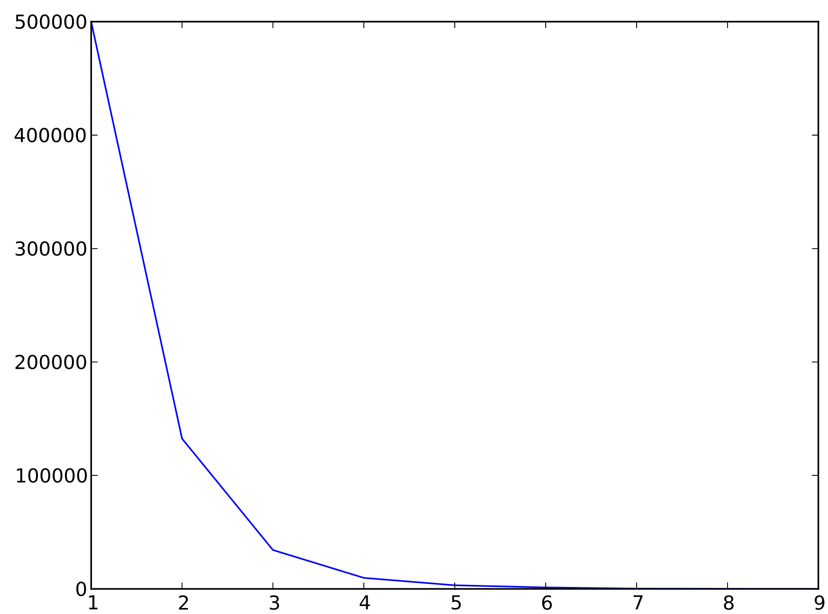

Figure 7: Number of vertices $|S_i|$ as a function of the number of steps.

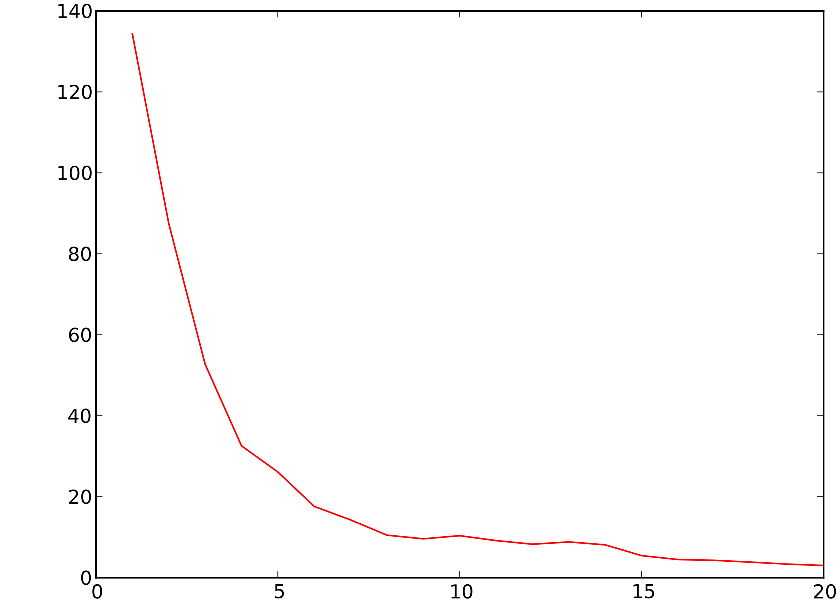

Figure 8: Densities $\rho(S_i)$ for each of 20 communities.

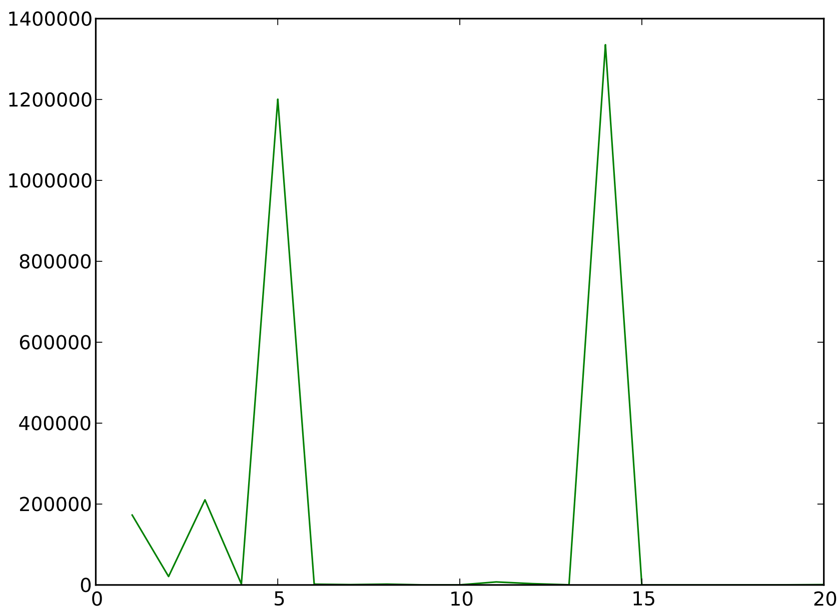

Figure 9: Edge sizes $|E[S_i]|$ for each of 20 communities.

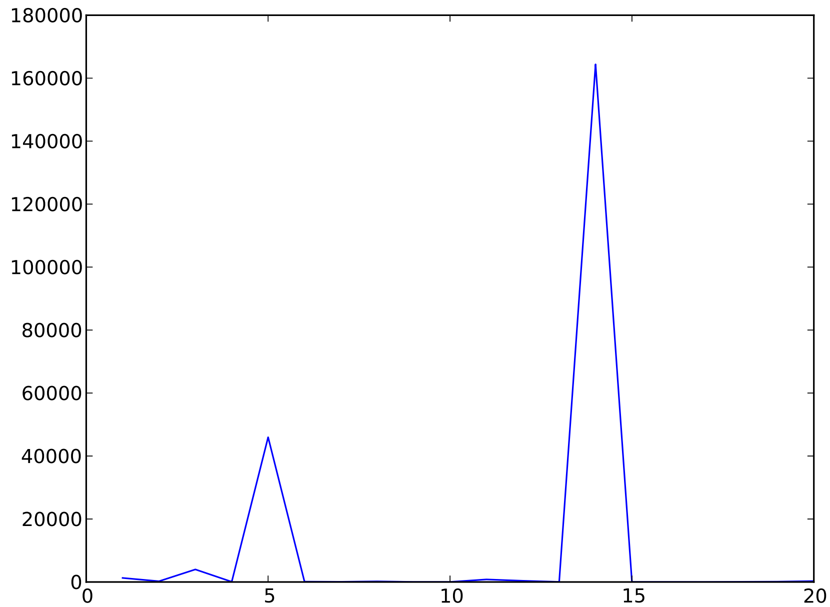

Figure 10: Number of vertices $|S_i|$ for each of 20 communities.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
#include <map>
#include <random>
#include <ctime>

class Params {
public:
  Params(const int& k_ = 20) : k(k_) { }

  void init(const int& m, const int& n); // pass the number of movies
  double dot(const int& qi, const int& pu) const;
  double all_norms() const {
    double norms = 0.0;
    for (auto it = p_idxs.begin(); it != p_idxs.end(); ++it) { norms += norm_p
        (it->second); }
    for (auto it = q_idxs.begin(); it != q_idxs.end(); ++it) { norms += norm_q
        (it->second); }
    return norms;
  }
  double norm_q(int qi) const { // square of L2 norm
    double norm = 0.0;
```

```
    for (int i = 0; i < k; ++i) { norm += Q[qi+i]*Q[qi+i]; }
    return norm;
  }
  double norm_p(int pi) const { // square of L2 norm
    double norm = 0.0;
    for (int i = 0; i < k; ++i) { norm += P[pi+i]*P[pi+i]; }
    return norm;
  }
  void update(const double* q, const int& qi, const double* p, const int& pi);
public:
  const int k;
  double* P;
  double* Q;
  std::map<int, int> p_idxs;
  std::map<int, int> q_idxs;
};

void Params::init(const int& m_, const int& n_) {
  std::default_random_engine generator((int)time(0));
  std::uniform_real_distribution<double> dist(0.0, sqrt(5.0/k));
  const int m = m_+1, n = n_+1;
  Q = new double[m*k];
  for (int i = 0; i < m*k; ++i) { Q[i] = dist(generator); }
  P = new double[n*k];
  for (int i = 0; i < n*k; ++i) { P[i] = dist(generator); }
}

double Params::dot(const int& qi, const int& pu) const {
  double s = 0.0;
  for (int j = 0; j < k; ++j) { s += Q[qi+j]*P[pu+j]; }
  return s;
}

void Params::update(const double* q, const int& qi, const double* p, const int& pi) {
  for (int j = 0; j < k; ++j) { Q[qi+j] = q[j]; P[pi+j] = p[j]; }
}

void sgd(const std::string& fname, Params& params, const int& k, const double& lambda, const int& iterations, const double& eta);
double estimate_error(const std::string &fname, Params& params, const int& k, const double& lambda);

// entry point
int main(int argc, char** argv) {
  Params params(20);
  sgd("ratings.train.txt", params, 20, 0.2, 40, 0.01565);
  return 0;
}

```

```
void sgd(const std::string& fname, Params& params, const int& k = 20, const double& lambda = 0.2, const int& iterations = 40, const double& eta = 0.1) {
  double crr_eps = 0.0, tmp_eps = 0.0;
  int p_idx = 0, q_idx = 0;
  int crr_user = 0, crr_movie = 0, crr_rating = 0;
  // preliminary pass to initialize params
  std::string line;
  std::ifstream infile(fname.c_str());
  getline(infile, line);
  do {
    int t = line.find_first_of('\t', 0);
    crr_user = atoi(line.substr(0, t).c_str());
    crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
    if (params.p_idxs.find(crr_user) == params.p_idxs.end()) { params.p_idxs[crr_user] = p_idx++; }
    if (params.q_idxs.find(crr_movie) == params.q_idxs.end()) { params.q_idxs[crr_movie] = q_idx++; }
    getline(infile, line);
  } while (!infile.eof());
  infile.close();

  params.init(q_idx-1, p_idx-1);

  double* p = new double[k]();
  double* q = new double[k]();

  // do k iterations of stohastic gradient descent
  for (int i = 0; i < iterations; ++i) {
    // for each r_{iu} in the data
    std::ifstream infile(fname.c_str());
    getline(infile, line);
    do {
      int t = line.find_first_of('\t', 0);
      crr_user = atoi(line.substr(0, t).c_str());
      crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
      crr_rating = atoi(line.substr(line.find_last_of('\t')+1).c_str());
      // compute epsilon
      p_idx = k*params.p_idxs[crr_user];
      q_idx = k*params.q_idxs[crr_movie];
      // update epsilon for next iteration
      crr_eps = crr_rating - params.dot(q_idx, p_idx);
      // now update the vectors q and p
      for (int j = 0; j < k; ++j) {
        q[j] = params.Q[q_idx+j] + eta*(crr_eps*params.P[p_idx+j] - lambda*params.Q[q_idx+j]);
        p[j] = params.P[p_idx+j] + eta*(crr_eps*params.Q[q_idx+j] - lambda*params.P[p_idx+j]);
        if (std::isnan(p[j]) || std::isnan(q[j])) { std::cout << "ISNAN: " << i << "-" << j << std::endl; }
      }
```

```
      params.update(q, q_idx, p, p_idx);

      getline(infile, line);
    } while (!infile.eof() && line.length() > 2);
    infile.close();
    std::cout << i << ":" << estimate_error(fname, params, k, lambda) << std::endl;
  }
  std::cout << "Using eta=" << eta << std::endl;
}

double estimate_error(const std::string& fname, Params& params, const int& k,
    const double& lambda) {
  double err = 0.0, crr_err = 0.0;
  std::ifstream infile(fname.c_str());
  std::string line;
  int crr_movie = 0, crr_user = 0, crr_rating = 0;
  int p_idx = 0, q_idx = 0;
  getline(infile, line);
  do {
    int t = line.find_first_of('\t', 0);
    crr_user = atoi(line.substr(0, t).c_str());
    crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
    crr_rating = atoi(line.substr(line.find_last_of('\t')+1).c_str());
    p_idx = k*params.p_idxs[crr_user];
    q_idx = k*params.q_idxs[crr_movie];
    crr_err = crr_rating - params.dot(q_idx, p_idx);
    err += crr_err*crr_err;
    getline(infile, line);
  } while (!infile.eof() && line.length() > 2);
  infile.close();
  err += lambda*(params.all_norms());
  return err;
}
```

Listing 2: SGD code for HW3Q1 item c.

```
//
// Compile with `g++ -std=c++0x -O3 sgd.cpp -o sgd'
// -- Blaz Sovdat, 18 feb 2014
//

#include <algorithm>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
#include <map>
#include <random>
#include <ctime>

```

```
class Params {
public:
  Params(const int& k_ = 20) : k(k_) { }

  void init(const int& m, const int& n); // pass the number of movies
  double dot(const int& qi, const int& pu) const;
  double all_norms() const {
    double norms = 0.0;
    for (auto it = p_idxs.begin(); it != p_idxs.end(); ++it) { norms += norm_p(k*it->second); }
    for (auto it = q_idxs.begin(); it != q_idxs.end(); ++it) { norms += norm_q(k*it->second); }
    return norms;
  }
  double norm_q(const int& qi) const { // square of L2 norm
    double norm = 0.0;
    for (int i = 0; i < k; ++i) { norm += Q[qi+i]*Q[qi+i]; }
    return norm;
  }
  double norm_p(const int& pi) const { // square of L2 norm
    double norm = 0.0;
    for (int i = 0; i < k; ++i) { norm += P[pi+i]*P[pi+i]; }
    return norm;
  }
  void update(const double *q, const int& qi, const double* p, const int& pi);
public:
  const int k;
  double* P;
  double* Q;
  std::map<int, int> p_idxs;
  std::map<int, int> q_idxs;
};

void Params::init(const int& m_, const int& n_) {
  std::default_random_engine generator((int)time(0));
  std::uniform_real_distribution<double> dist(0.0, sqrt(5.0/k));
  const int m = m_+1, n = n_+1;
  Q = new double[m*k];
  for (int i = 0; i < m*k; ++i) { Q[i] = dist(generator); }
  P = new double[n*k];
  for (int i = 0; i < n*k; ++i) { P[i] = dist(generator); }
}

double Params::dot(const int& qi, const int& pu) const {
  double s = 0.0;
  for (int j = 0; j < k; ++j) { s += Q[qi+j]*P[pu+j]; }
  return s;
}

void Params::update(const double* q, const int& qi, const double* p, const int& pi) {
```

```
  for (int j = 0; j < k; ++j) { Q[qi+j] = q[j]; P[pi+j] = p[j]; }
}

void sgd(const std::string& fname, Params& params, const int& k, const double& lambda, const int& iterations, const double& eta);
double estimate_error(const std::string& fname, Params& params, const int& k, const double& lambda);

// entry point
int main(int argc, char** argv) {
  for (int k = 1; k <= 10; ++k) {
    Params params(k);
    // sgd("ratings.train.txt", params, k, 0.0, 40, 0.03);
    sgd("ratings.train.txt", params, k, 0.2, 40, 0.03);
    // std::cout << k << ":" << estimate_error("ratings.val.txt", params, k, 0.0) << std::endl;
    std::cout << k << ":" << estimate_error("ratings.val.txt", params, k, 0.2) << std::endl;
  }
  return 0;
}

void sgd(const std::string& fname, Params& params, const int& k = 20, const double& lambda = 0.2, const int& iterations = 40, const double& eta = 0.1) {
  double crr_eps = 0.0, tmp_eps = 0.0;
  int p_idx = 0, q_idx = 0;
  int crr_user = 0, crr_movie = 0, crr_rating = 0;
  // preliminary pass to initialize params
  std::string line;
  std::ifstream infile(fname.c_str());
  getline(infile, line);
  do {
    int t = line.find_first_of('\t', 0);
    crr_user = atoi(line.substr(0, t).c_str());
    crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
    // set indices
    if (params.p_idxs.find(crr_user) == params.p_idxs.end()) { params.p_idxs[crr_user] = p_idx++; }
    if (params.q_idxs.find(crr_movie) == params.q_idxs.end()) { params.q_idxs[crr_movie] = q_idx++; }
    getline(infile, line);
  } while (!infile.eof());
  infile.close();
  // initialize matrices P and Q
  params.init(q_idx-1, p_idx-1);
  // temporary k-dimensional vectors
  double* p = new double[k]();
  double* q = new double[k]();

  // do k iterations of stohastic gradient descent
```

```
  for (int i = 0; i < iterations; ++i) {
    // for each r_{iu} in the data
    std::ifstream infile(fname.c_str());
    getline(infile, line);
    do {
      int t = line.find_first_of('\t', 0);
      // retrieve indices
      crr_user = atoi(line.substr(0, t).c_str());
      crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
      crr_rating = atoi(line.substr(line.find_last_of('\t')+1).c_str());
      // compute epsilon
      p_idx = k*params.p_idxs[crr_user];
      q_idx = k*params.q_idxs[crr_movie];
      // update epsilon for next iteration
      crr_eps = crr_rating - params.dot(q_idx, p_idx);
      // now update the vectors q and p
      for (int j = 0; j < k; ++j) {
        q[j] = params.Q[q_idx+j] + eta*(crr_eps*params.P[p_idx+j] - lambda*
            params.Q[q_idx+j]);
        p[j] = params.P[p_idx+j] + eta*(crr_eps*params.Q[q_idx+j] - lambda*
            params.P[p_idx+j]);
        if (std::isnan(p[j]) || std::isnan(q[j])) { std::cout << "ISNAN: " <<
            i << "-" << j << std::endl; }
      }
      // assign them to parameters matrices
      params.update(q, q_idx, p, p_idx);

      getline(infile, line);
    } while (!infile.eof() && line.length() > 2);
    infile.close();
  }
  std::cout << "Error for k=" << k << ": " << estimate_error(fname, params, k,
      lambda) << std::endl;
}

double estimate_error(const std::string& fname, Params& params, const int& k,
    const double& lambda) {
  double err = 0.0, crr_err = 0.0;
  std::ifstream infile(fname.c_str());
  std::string line;
  int crr_movie = 0, crr_user = 0, crr_rating = 0;
  int p_idx = 0, q_idx = 0;
  getline(infile, line);
  do {
    int t = line.find_first_of('\t', 0);
    crr_user = atoi(line.substr(0, t).c_str());
    crr_movie = atoi(line.substr(t+1, line.find_last_of('\t')).c_str());
    crr_rating = atoi(line.substr(line.find_last_of('\t')+1).c_str());
    if (params.p_idxs.find(crr_user) != params.p_idxs.end() && params.q_idxs.
        find(crr_movie) != params.q_idxs.end()) {
      p_idx = k*params.p_idxs[crr_user];
```

```
      q_idx = k*params.q_idxs[crr_movie];
      crr_err = crr_rating - params.dot(q_idx, p_idx);
      err += crr_err*crr_err;
    }
    getline(infile, line);
  } while (!infile.eof() && line.length() > 2);
  infile.close();
  // err += lambda*(params.all_norms()); // error function was redefined for
      this bullet
  return err;
}
```

A.2 PageRank Computation

Listing 3: PageRank code for HW3Q2.

```
#!/usr/bin/python

import scipy as sp
from scipy.sparse import *
import numpy as np
from scipy import *
import random as rnd
from time import time

avg = lambda L: 1.0*sum(L)/len(L)

def readfile(fname):
  for edge in open(fname):
    yield [int(v) for v in edge.split('\t')]

def error(P, R, n):
  IP = sorted(range(len(P)), key = lambda k: P[k])
  IR = sorted(range(len(R)), key = lambda k: R[k])
  IP.reverse()
  IR.reverse()
  return 1.0*sum([abs(P[IP[i]] - R[IR[i]]) for i in range(n)])/n

def mc(fname, R, beta = 0.8):
  G = {}
  for T in readfile(fname):
    if T[0] not in G: G[T[0]] = [T[1]]
    else: G[T[0]].append(T[1])

  n = len(G)
  # frequency count
  f = [0 for k in range(n)]
  # simulate random walk
  # for each vertex
  for v in G:
```

```
    # simulate it R times
    for iter in range(R):
      # walk terminates at each node with prob. 1-beta
      node = v
      while True:
        f[node-1] = f[node-1]+1
        if rnd.uniform(0, 1) > 1-beta: # continue the walk
          node = G[node][rnd.randint(0, len(G[node])-1)]
        else: # game over
          break
    # print f
  r = [(1-beta)*f[i]/(n*R) for i in range(n)]
  return r

def pagerank(fname, beta = 0.8):
  row = []
  col = []
  for T in readfile(fname):
    row.append(T[0]-1)
    col.append(T[1]-1)
  n = 100
  row = array(row)
  col = array(col)
  dat = array([1.0 for k in row])
  # convert it to full matrix to simplify our life
  M = csc_matrix((dat, (row, col)), shape = (n, n) ).todense()
  # make it column stohastic
  for i in range(n): M[:, i] =M[:, i].dot(1.0/M[:, i].sum())
  # initialize page rank vector
  r = np.transpose(array([1.0/n for k in range(n)]))
  t = np.transpose(array([(1-beta)/n for k in range(n)]))
  # now do power iteration
  for i in range(40):
    r = array(t+beta*M.dot(r))
    r = r[0,:]
  return r

# entry point
if __name__ == '__main__':
  # compute PageRank with Power iteration
  pr = pagerank('graph.txt', 0.8)
  #T = []
  #for i in range(50):
  # start = time()
  # pr = pagerank('graph.txt', 0.8)
  # T.append(time() - start)
  #print avg(T)

  # now run the MC algortihm
  r1 = mc('graph.txt', 1, 0.8)
  r3 = mc('graph.txt', 3, 0.8)
```

```
  r5 = mc('graph.txt', 5, 0.8)

  print "*"*3, "R = 1", "*"*3
  print "Top 10: ", error(pr, r1, 10)
  print "Top 30: ", error(pr, r1, 30)
  print "Top 50: ", error(pr, r1, 50)
  print "All   : ", error(pr, r1, len(r1))

  print "*"*3, "R = 3", "*"*3
  print "Top 10: ", error(pr, r3, 10)
  print "Top 30: ", error(pr, r3, 30)
  print "Top 50: ", error(pr, r3, 50)
  print "All   : ", error(pr, r3, len(r3))

  print "*"*3, "R = 5", "*"*3
  print "Top 10: ", error(pr, r5, 10)
  print "Top 30: ", error(pr, r5, 30)
  print "Top 50: ", error(pr, r5, 50)
  print "All   : ", error(pr, r5, len(r5))
```

A.3 Dense Communities in Networks

Listing 4: Streaming implementatoin of the algorithm from HW3Q4.

```
#!/usr/bin/python

from time import time
import matplotlib.pyplot as plt
import math

error = lambda P, R, I, n: 1.0*sum([abs(P[I[i]] - R[I[i]]) for i in range(n)])
    /n
avg = lambda L: 1.0*sum(L)/len(L)

# we count each edge twice, 2|E(G)| = sum_v deg(v), so we need to factor out 2
density = lambda S: avg(S.values())/2 if len(S) > 0 else 0

def readfile(fname):
  for edge in open(fname):
    yield [int(v) for v in edge.split()]

# basic version of the algorithm
def algo(fname, eps):
  S = {}
  T = {}
  for e in readfile(fname):
    # print e
    S[e[0]] = 1 if e[0] not in S else S[e[0]]+1
    S[e[1]] = 1 if e[1] not in S else S[e[1]]+1
  dS = density(S)
```

```
    print dS
    i = 1
    plot_rhos = [dS]
    plot_edgs = [sum(S.values())/2]
    plot_sizs = [len(S)]
    plot_X = [i]
    while len(S) > 0:
        tS = {}
        for v in S:
            if S[v] > 2*(1+eps)*dS: tS[v] = 0
        for e in readfile(fname): # compute S \ A(S)
            if e[0] in tS and e[1] in tS:
                tS[e[0]] = tS[e[0]]+1
                tS[e[1]] = tS[e[1]]+1
        S = dict(tS)
        dS = density(tS)
        if dS > density(T): T = dict(tS)
        i = i+1
        plot_rhos.append(dS)
        plot_edgs.append(sum(S.values())/2 if len(S) > 0 else 0)
        plot_sizs.append(len(S))
        plot_X.append(i)
        print "Current density: ", dS
        print "|S|=", len(S), "|T|=", len(T)
        print "-"*80
    # print T
    print density(T)
    print "Done"

    # print "rhos=", plot_rhos
    # print "edgs=", plot_edgs
    # print "sizs=", plot_sizs

    plt.plot(plot_X, plot_rhos, 'r')
    plt.draw()

    plt.figure()
    plt.plot(plot_X, plot_edgs, 'g')
    plt.draw()

    plt.figure()
    plt.plot(plot_X, plot_sizs, 'b')
    plt.draw()

    plt.show()

    return i

# stops after it finds 20 communities
# NOTE: Due to time constraints the code is not efficient
def multiple_components(fname, eps = 0.05):
```

```
  D = set()
  plot_rhos = []
  plot_sizs = []
  plot_edgs = []
  plot_X = []
  for i in range(1, 21):
    start = time()
    S = {}
    T = {}
    for e in readfile(fname):
      if e[0] not in D and e[1] not in D:
        S[e[0]] = 1 if e[0] not in S else S[e[0]]+1
        S[e[1]] = 1 if e[1] not in S else S[e[1]]+1
    dS = density(S)
    while len(S) > 0:
      tS = {}
      for v in S:
        if S[v] > 2*(1+eps)*dS: tS[v] = 0
      for e in readfile(fname): # compute S \ A(S)
        if e[0] in tS and e[1] in tS:
          tS[e[0]] = tS[e[0]]+1
          tS[e[1]] = tS[e[1]]+1
      S = dict(tS)
      dS = density(tS)
      if dS > density(T): T = dict(tS)
    plot_rhos.append(density(T))
    plot_edgs.append(sum(T.values())/2 if len(T) > 0 else 0)
    plot_sizs.append(len(T))
    plot_X.append(i)
    for v in T: D.add(v)
    print "Current density: ", density(T)
    print "|T|=", len(T)
    print "|E[T]|=", sum(T.values())/2
    print "Community found in ", time()-start, "seconds"
    print "-"*80

  plt.plot(plot_X, plot_rhos, 'r')
  plt.draw()

  plt.figure()
  plt.plot(plot_X, plot_edgs, 'g')
  plt.draw()

  plt.figure()
  plt.plot(plot_X, plot_sizs, 'b')
  plt.draw()

  plt.show()

# entry point
if __name__ == '__main__':
```

```
# algo('livejournal-undirected-small.txt', 0.01)
"""
n = 499923
X = [0.1, 0.5, 1.0, 2.0]
for eps in X:
    start = time()
    # algo returns the number of steps
    Y.append(algo('livejournal-undirected.txt', eps))
    print "time: ", time()-start, "seconds"
print "X=", X
print "Y=", Y
plt.plot(X, Y)
plt.plot(X, [math.log(n, 1+x) for x in X])
plt.show()
"""

# print algo('livejournal-undirected.txt', 0.05)

multiple_components('livejournal-undirected.txt', 0.05)
```

References