CS 395T	Scalable Machine Learning	Fall 2014

Solutions to Homework 4

Lecturer: Inderjit Dhillon — Date Due: Sep 29, 2014

Keywords: *Sparse matrix, Incremental k-means*

Problem 1

(a) k-means objective is to maximize:

$$\sum_{i=1}^{k} \sum_{j:\pi(j)=i} \boldsymbol{c}_i^T \boldsymbol{x}_j = \sum_{i=1}^{k} \boldsymbol{c}_i^T \Big(\sum_{j:\pi(j)=i} \boldsymbol{x}_j \Big) \equiv J.$$

Using the fact that $\boldsymbol{s}_i = \sum_{j:\pi(j)=i} \boldsymbol{x}_j$ and $\boldsymbol{c}_i = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$, we have $J = \sum_{i=1}^{k} \frac{\boldsymbol{s}_i^T \boldsymbol{s}_i}{\|\boldsymbol{s}_i\|_2} = \sum_{i=1}^{k} \|\boldsymbol{s}_i\|_2$.

(b) Let $\tilde{q}_{\pi(i)}$ denote the quality of cluster $\pi(i)$ when $\boldsymbol{x}_i$ is removed from its cluster $\pi(i)$. We can write:

$$\tilde{q}_{\pi(i)} = \left(q_{\pi(i)}^2 - 2 q_{\pi(i)} S(i, \pi(i) + 1 \right)^{1/2}.$$

Similarly, let $\tilde{q}_j$ denote the quality of the cluster j after $\boldsymbol{x}_i$ is assigned to cluster j. We can write:

$$\tilde{q}_j = \left(q_j^2 + 2 q_j S(i, j) + 1 \right)^{1/2}.$$

Notice that $\Delta^{(j,i)} = Q^{(j,i)} - Q^{(t)} = \tilde{q}_{\pi(i)} + \tilde{q}_j - (q_{\pi(i)} + q_j)$, as the other cluster qualities remain unchanged. Each of the quantities can be evaluated in $O(1)$ and $\Delta^{(j,i)}$ itself can be computed in $O(1)$.

(c) At the end of an iteration, only 2 clusters potentially change $\pi(i^*)$ and j^*. We can update the centers of these clusters in $O(d)$. This requires updating the corresponding two columns in matrix S which can be done in $O(Nd)$. Updating π is just setting $\pi(i^*) = j^*$. Finally, we can update the qualities of the 2 clusters in time $O(d)$.

Problem 2

(a) Below is the header file that contains the class for the sparse matrix. It implements the required parallel sparse matrix-vector multiplication `y = A.multiply(x)`.

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
#include <vector>
#include <cmath>
#include <omp.h>
#include <assert.h>
```

```
#define MALLOC(type, size) (type*)malloc(sizeof(type)*(size))

// Sparse matrix format RCS
class smat_t{
    public:
        long rows, cols;
        long nnz;
        double *val;
        long *row_ptr;
        unsigned *col_idx;
        bool mem_alloc_by_me;
        smat_t():mem_alloc_by_me(false){ }
        smat_t(smat_t& m){ *this = m; mem_alloc_by_me = false;}
        void load(long _rows, long _cols, long _nnz, const char*filename) {
            rows =_rows,cols=_cols,nnz=_nnz;
            mem_alloc_by_me = true;
            val = MALLOC(double, nnz);
            col_idx = MALLOC(unsigned, nnz);
            row_ptr = MALLOC(long, rows+1);
            memset(row_ptr,0,sizeof(long)*(rows+1));

            FILE *fp = fopen(filename, "r");
            // Read row-format data into CSR matrix
            for(long i=0,r,c; i<nnz; ++i){
            fscanf(fp,"%ld %ld %lf", &r, &c, &val[i]);
            row_ptr[r]++;
            col_idx[i] = c-1;
        }
        fclose(fp);
        for(long r=1; r<=rows; ++r) row_ptr[r] += row_ptr[r-1];
        }
        long nnz_of_row(int i) const {return (row_ptr[i+1]-row_ptr[i]);}
        void free(void *ptr) {if(!ptr) ::free(ptr);}
        ~smat_t(){
            if(mem_alloc_by_me) {
                    if(val)free(val);
                    if(row_ptr)free(row_ptr);
                    if(col_idx)free(col_idx);
            }
        }
        void clear_space() {
            if(val)free(val);
            if(row_ptr)free(row_ptr);
            if(col_idx)free(col_idx);
            mem_alloc_by_me = false;
        }
        double* multiply (double *x) {
```

```
            double *y = (double *) malloc(rows * sizeof(double));
            #pragma omp parallel for schedule(dynamic,32)
            for (long i=0; i < rows; i++) {
                double v=0.0;
                for (long j=row_ptr[i]; j < row_ptr[i+1]; j++) {
                    v += val[j]*x[col_idx[j]];
                }
                y[i] = v;
            }
            return y;
        }
};
```

(b) Runtimes, in seconds, for #threads $1, 4, 8, 16$ respectively are 0.9092, 0.2496, 0.1316 and 0.09850.

Problem 2

(a) The time taken for the C++ code using 16 threads and $T = 50$ is 11.0222 seconds. Corresponding Matlab implementation takes about 35 seconds.

(b) Note that $\lambda = \frac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}} = \boldsymbol{x}^T A \boldsymbol{x}$, as $\boldsymbol{x}^T \boldsymbol{x} = 1$. After 50 iterations, the converged λ for the given sparse matrix A is 663.3333.

(c) The top 100 nodes are given below:

Rank	Node index	Rank	Node index
1	17634	51	1220958
2	17596	52	1374951
3	17602	53	17706
4	17616	54	1516650
5	17610	55	17639
6	17779	56	17637
7	17705	57	17893
8	17595	58	17594
9	17585	59	17592
10	17615	60	17774
11	17647	61	17590
12	17831	62	17576
13	17582	63	17669
14	17631	64	1177530
15	17760	65	17739
16	17577	66	17678
17	17618	67	17562
18	17697	68	17795
19	17658	69	17765
20	17703	70	17677
21	17619	71	17766
22	68822	72	17817
23	17606	73	1418548
24	17581	74	17793
25	17646	75	17717
26	17593	76	17714
27	17609	77	17720
28	17662	78	17865
29	17796	79	17684
30	17572	80	17683
31	17591	81	1363911
32	17680	82	1161724
33	17627	83	17587
34	17659	84	17758
35	1515012	85	1217996
36	17621	86	17902
37	17598	87	17740
38	17578	88	17605
39	17574	89	17588
40	17635	90	17833
41	17599	91	17792
42	1363532	92	17768
43	17660	93	1467533
44	17644	94	68903
45	17689	95	18021
46	1418209	96	1492397
47	904683	97	180035
48	17724	98	17823
49	1488543	99	17731
50	545289	100	17818