CS 395T	Scalable Machine Learning	Fall 2014
	Homework 4	
Lecturer: Inderjit Dhillon		Date Due: Oct 6, 2014
Keywords: *Sparse matrix, Incremental k-means*		

Note:

1. A download link to the dataset (`hw4-data.mat`) is given in the Assignments tab in Canvas.
2. Submit your code for Problems 2 and 3 through Canvas. Submit a hard copy of your answers in class.

Problem 1: Incremental k-means Clustering

The k-means algorithm with cosine similarity, also known as the batch k-means algorithm, is a popular method for clustering document collections. However, batch k-means can often yield qualitatively poor results, especially for small clusters, say 25-30 documents per cluster, where it tends to get stuck at a local maximum far away from the optimal. The *incremental* k-means strategy overcomes the drawback of batch k-means — the idea is to refine a given clustering by incrementally moving data points between clusters, thus achieving a higher objective function value.

Let $\mathbf{x}_i \in \mathbb{R}^d,\ i = 1, 2, \ldots, N$ denote the document vectors. Let k denote the desired number of clusters. Assume $\mathbf{x}_i$ is normalized such that $\|\mathbf{x}_i\|_2 = 1$. Let $\pi(i)$ denote the cluster assignment of $\mathbf{x}_i$. For each cluster k, let $\mathbf{s}_k = \sum_{i:\pi(i)=k} \mathbf{x}_i$. The normalized centroid of cluster k is then given by $\mathbf{c}_k = \frac{\mathbf{s}_k}{\|\mathbf{s}_k\|_2}$. The objective function that we want to *maximize* is

$$Q = \sum_{z=1}^{k} \|\mathbf{s}_z\|_2\ . \tag{1}$$

For each document $\mathbf{x}_i$, let $\pi^{(t)}(i)$ denote its current cluster assignment. At iteration t, incremental k-means algorithm computes the following quantity called *gain*, denoted by $\Delta^{(j,i)}$, for each point $\mathbf{x}_i$ and each cluster j:

$$\Delta^{(j,i)} = Q^{(j,i)} - Q^{(t)}, \tag{2}$$

where $Q^{(j,i)}$ denotes the objective value of the partitioning obtained by *removing* $\mathbf{x}_i$ from the cluster $\pi^{(t)}(i)$ and *assigning* $\mathbf{x}_i$ to cluster j, and $Q^{(t)}$ is the objective value for the current partitioning. The algorithm then makes the best possible assignment, i.e. assigns $\mathbf{x}_i^*$ to cluster j^*, where j^*, i^* maximize $\Delta^{(j,i)}$.

(a) (2 pt) Show that (1) is the k-means objective using cosine similarity. Cosine similarity between two vectors $\mathbf{x}$ and $\mathbf{y}$ is defined as $\frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$.

(b) (4 pt) Computing the gain $\Delta^{(j,i)}$ as given in (2) can take $O(d)$ time, where m is the dimensionality of the data. Assume you are given (i) similarity matrix S, where $S_{i,j}$ is the cosine similarity between $\mathbf{x}_i$ and cluster centroid $\mathbf{c}_j$, (ii) vector q where $q_k = \|\mathbf{s}_k\|_2$ is the quality of cluster k, (iii) cluster assignments π. Show that you can compute the gain (2) in $O(1)$ time.

(c) (2 pt) How would you efficiently update the quantities (i), (ii) and (iii) given in part (b) after each iteration?

Problem 2: Sparse Matrix-Vector Multiplication

In this problem, you will implement sparse matrix-vector multiplication in C/C++ and parallelize it. Design a sparse matrix data structure to efficiently store and access the non-zeros of the matrix.

(a) (8 pt) Implement the function `y = A.multiply(x)` which performs parallel sparse matrix-vector multiplication.

(b) (4 pt) Using the sparse matrix A in `hw4-data.mat` and random vectors $\mathbf{x}$, report (averaged) running times for 1, 4, 8 and 16 threads.

Note: We have given you the matrix A as a Matlab sparse matrix. You should convert it into text format to be loaded from your C/C++ code, in row-major or column-major order as per your requirements.

Problem 3: Eigenvector centrality

In this problem, you will write C/C++ code to compute the Eigenvector centrality, which is a popular measure of centrality of nodes in social networks (other measures of centrality include *degree* centrality and *betweenness* centrality). The Eigenvector centrality measure of a node u, whose neighbors in the network are denoted by $\mathcal{N}(u)$, is defined in terms of the measures of the nodes to which it is connected — a central node should be one connected to powerful nodes.

$$x_u = \frac{1}{\lambda} \sum_{v \in \mathcal{N}(u)} x_v.$$

where λ is the dominant eigenvalue of the network adjacency matrix. The above equation can be written as the Eigenvector equation: $A\mathbf{x} = \lambda\mathbf{x}$, where A denotes the adjacency matrix of the network. We employ the power iterations method to compute the Eigenvector centrality. The algorithm is given below:

1. Init $\mathbf{x}^0 \leftarrow \mathbf{1}$, the vector of all ones.
2. $\mathbf{x}^0 \leftarrow \mathbf{x}^0 / \|\mathbf{x}^{(0)}\|_2$.
3. For $t = 1, 2, \ldots, T$
   (a) Update $x_i^{(t+1)} \leftarrow \sum_{j:A_{ij} \neq 0} A_{ij} x_j^t$.
   (b) $\mathbf{x}^{(t+1)} \leftarrow \frac{\mathbf{x}^{(t+1)}}{\|\mathbf{x}^{(t+1)}\|_2}$.

Use the sparse matrix-vector multiply solver developed in Problem 2 and implement the above algorithm in C/C++.

1. (2 pt) Run your code for the (undirected) network A in `hw4-data.mat` and report the running times for 16 threads and $T = 50$. Compare the runtime with the corresponding Matlab implementation (with the same T).
2. (1 pt) How will you obtain the converged eigenvalue λ at the end of 50 iterations (using $A\mathbf{x} = \lambda\mathbf{x}$)? Report the converged λ for the matrix A.
3. (2 pt) List the top 100 nodes of A with the highest Eigenvector centrality measure — give a table with two columns Rank and Node Index (starting with 1).