Rutgers CS323 (04), Spring 2017, Homework 1

Due at 8:00am on Feb 13, 2017, submitted via Sakai

K-Means Clustering

1 Introduction

The K-means algorithm is the simplest and most widely used method for clustering, in an iterative fashion. Given n data points $\{x_j\}_{j=1}^n$ in p dimensions (i.e., $x_j \in \mathbb{R}^p$), the general goal of clustering is to partition the n data points into K groups (clusters), denoted by $\{C_1, C_2, ..., C_K\}$, such that the following objective function is (hopefully) minimized:

$$SD = \sum_{k=1}^{K} \sum_{x_j \in C_k} \|x_j - \mu_k\|^2 \tag{1}$$

where $\mu_k = \frac{1}{|C_k|} \sum_{j \in C_k} x_j$ is the center of the k-th cluster and

$$\|x_j - \mu_k\|^2 = \sum_{s=1}^{p} |x_{j,s} - \mu_{k,s}|^2$$

is the squared Euclidean distance between x_j and μ_k.

The objective (1), however, is not a convex problem. For high-dimensional data (i.e., large p), in general one can only expect to obtain approximate (sub-optimal) solutions.

K-means is an iterative algorithm. At the beginning, K data points from $\{x_j\}_{j=1}^n$ are randomly selected as the clustering center. In each iteration, we compute the squared Euclidean distance between each data point and each cluster center, and we assign a data point to the cluster whose center is the nearest (among K centers). Once all data points are assigned to clusters, we re-compute the cluster centers by simple averaging. The iteration continues till some convergence condition is met or the maximum number of iterations is reached.

The K-means algorithm is conceptually simple and its basic version is easy to implement. One can easily modify the algorithm for special needs. For example, for some applications, the squared Euclidean distance may not be as suitable as other distance/similarity measures. In fact, users can define their own distance/similarity metrics.

2 Implementation

You will need to implement two functions:

```
function TestMyKmeans(filename,numRepeat, numIter)
function [idx,C,sumD,D] = MyKmeans(X,K,C0,numIter)
```

where $X \in \mathbb{R}^{n \times p}$ is the n by p data matrix, K is number of clusters, $C0 \in \mathbb{R}^{K \times p}$ are the initial cluster centers, "numIter" is a pre-specified number of iterations. "numRepeat" is the number of times we want to repeat the experiments. The following two executions produced Figure 1 and Figure 2:

```
TestMyKmeans('zip.train',7,60);
TestMyKmeans('text.mat',7,60);
```

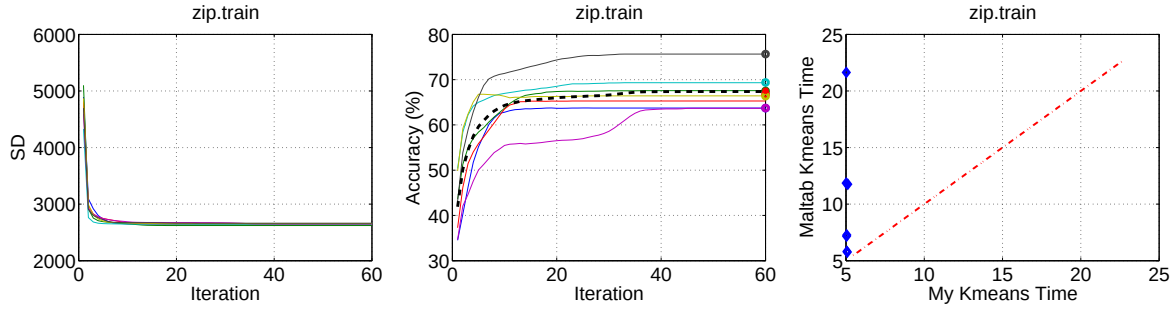

Figure 1: Experimental results on **zip.train**. The **leftmost panel** plots the SD (1) values against iterations, verifying that the K-means procedure indeed decrease the desired objective function, with diminishing returns as the number of iterations increases. Note that there are **7** curves because "numRepeat" is set to be 7 in this run and each run uses a different set of starting points. The **middle panel** plots the classification accuracies against iterations. Note that the K-means procedure does not use any information about the class labels. Nevertheless it is a convenient and intuitive way to measure the clustering results by classifications since the labels are known anyway. The code will be provided to evaluate the accuracies. Note that in addition to 7 curves, there is also an **averaging** curve. The 7 small circles (and the star for the average) at the last iteration are the results from **Matlab K-means**. The **rightmost panel** compares the times with the Matlab build-in implementation. It turns out that Maltab does not use an efficient implementation of K-Means. For this assignment, you don't have to worry too much about efficiency. It is just fun to plot the times.

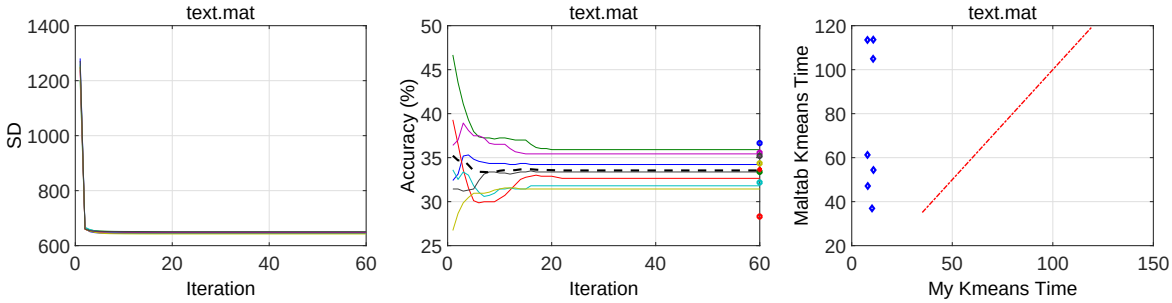

Figure 2: Experimental results on **text.mat**. This is a very sparse text data. It turns out Matlab K-means does not optimize the code for sparse data hence it uses substantially more times than my code. But again, for this homework, you don't have to worry about the times.

For your convenient, a significant portion of the testing code is posted as follows:

```
function TestMyKmeans(filename,numRepeat, numIter)

data = importdata(filename);
Y = data(:,1)+1; X = data(:,2:end); clear data;
K = max(Y);   n = length(Y);

%%%%%%
%%% Normalize the data to have unit L2 norm
%%%%

SD1 = zeros(numRepeat,numIter);
for i = 1:numRepeat;
    i
    C0 = X(randsample(n,K),:);
    tic;[idx1{i},C1,SD1(i,:),D1{i}]=MyKmeans(X,K,C0,numIter); T1(i) = toc;
    tic;[idx2{i},C2,sumd2,D2{i}]=kmeans(full(X),K,'Start',full(C0),'Maxiter',numIter)
    T2(i)=toc;
    SD2(i,:) = sum(sumd2);
    for t = 1:numIter;
        acc1(i,t) = evalClust_Error(idx1{i}(t,:),Y);
    end
```

```
    %%%%%%%
    %%%% Evluate the classification accuracy for Maltab Kmeans algorithm
    %%%%%%%%

    output = [acc1(1:i,end) acc2(1:i)' SD1(1:i,end) SD2(1:i) T1(1:i)' T2(1:i)'];
    feval('save',[filename '.summary.txt'],'output','-ascii');
end

figure;
plot(1:numIter,SD1,'linewidth',1);hold on; grid on;
set(gca,'FontSize',20);
xlabel('Iteration');ylabel('SD');
title(filename);

figure;
%%%%%%
%%% Plot accuracies
%%%%
figure;
%%%%%%%%
%%%% Plot times
%%%%%%%%
```

To evaluate the classification accuracies, you will need to use two additional functions provided in the sample code repository:

```
function evalClust_Error
function hungarian
```

3 Several Issues

You will need to normalize the data to have unit $L2$ norm at the beginning of the process. Basically, this means, for every row in the data matrix, x_j, we have $\sum_{i=1}^{p} |x_{j,i}|^2 = 1$. It turns out for K-means using Euclidian distance, the normalization step is actually quite important.

You will learn from the code how to measure the execution time for any chunk of the code, which might be very useful in your later work (after this class), a small bonus for taking this course. For this assignment, however, you are not required to beat Maltab (although it is a fun thing to try).

The text.mat dataset is a sparse text dataset. You are not required to do anything special (since the dataset will be loaded a sparse matrix anyway),

although you might want to keep that in mind and try to understand why Matlab is so slow for K-means.

It turns out that Matlab K-means does not always produce exactly the same outputs as our code, although the difference is usually very small (if any and becomes negligible once we average over several runs).

4 Submission Instruction

Your submission should include **10** files with the following names:

- 2 matlab files:

  ```
  [1] MyKmeans.m
  [2] TestMyKmeans.m
  ```

- 2 results (text) files:

  ```
  [3] zip.train.summary.txt
  [4] text.mat.summary.txt
  ```

- 3 figure files similar to Figure 1, using the same parameters.

  ```
  [5] zip_SD.fig
  [6] zip_acc.fig
  [7] zip_time.fig
  ```

- 3 figure files similar to Figure 2, using the same parameters.

  ```
  [8] text_SD.fig
  [9] text_acc.fig
  [10] text_time.fig
  ```

All the files will be submitted to Sakai before the deadline, in one zipped file (using WinZip).