CSE 847 Home Assignment 5

Submitted by: Ritam Guha (MSU ID: guharita)

Date: April 21, 2021

1 Clustering: K-Means

1. Elaborate the relationship between k-means and spectral relaxation of k-means. Is it possible that we obtain exact k-means solution using spectral relaxed k-means?

   Response: To discuss this issue, the rows of data matrix are considered to be the samples and columns are considered to be the features. Following the objective of k-means to reduce the sum-of-squared error (SSE), we can represent the same objetctive in a different format by considering each cluster as a set Π_j where j represent the cluster label and Π_j contains all the data points in the j^{th} cluster. The SSE can then be represented as:

   $SSE(\Pi_j) = \sum_{v \in \Pi_j} ||x_v - c_j||^2$

   where c_j is the cluster center of the corresponding cluster. The entire SSE can be found after summing over all the clusters.

   After some transformations in matrix form, the objective function can be represented as:

   $$SSE = \sum_{j=1}^{k} (trace(X_j^T X_j) - \frac{e^T}{\sqrt{n_j}} X_j^T X_j \frac{e}{\sqrt{n_j}}) \tag{1}$$

   This expression could be further simplified as:

   $$SSE = trace(X_j^T X_j) - trace(Y^T X_j^T X_j Y) \tag{2}$$

   where $Y = \left\{ \begin{array}{cccc} \frac{e}{\sqrt{(n_1)}} & 0 & . \; . & 0 \\ 0 & \frac{e}{\sqrt{(n_1)}} & . \; . & 0 \\ & & . & \\ & & . & \\ 0 & 0 & . \; . & \frac{e}{\sqrt{(n_K)}} \end{array} \right\}$ which is an $(n \times k)$ orthogonal matrix.

   The goal of the approach then becomes to minimize SSE by maximizing $trace(Y^T X_j^T X_j Y)$. The spectral relaxation of k-means is inspired from the fact that instead of using this specific expression for Y, it is possible to use any arbirary orthogonal matrix for Y. This leads to the relaxed maximization problem:

   $$max_{Y^T Y = I_k} trace(Y^T X_j^T X_j Y) \tag{3}$$

The first k vectors in the left singular matrix of X can produce the Y^* that maximizes this expression. Finally, the clusters can be found using regular k-means on Y^*. That's how spectral version of k-means works. As evident from this discussion, we can see that both k-means and spectral k-means are trying to minimize the same error function, but spectral k-means is first trying to project the dataset to a lower dimensional space which makes it easier to capture the clustering structures.

The spectral-relaxed k-means become completely equivalent to k-means when the expression for Y becomes equal to the matrix mentioned in Equation 2.

2. Implementation of k-means. Submit all the source code to D2L along with a short report on your observation.

   - Implement the k-means in Matlab using the alternating procedure introduced in the class (you will not get the credit if you use the build-in kmeans function in Matlab).
   - Implement the spectral relaxation of k-means. Create a random dataset and compare the k-means and spectral relaxed k-means.

   Response: In the previous part, the main concepts behind the spectral k-means was discussed. In both the cases, the regular k-means is used as the underlying clustering algorithm. It works using Block Coordinate Descend (BCD) minimizing one variable at a time by keeping the remaining ones fixed. The implementation of the algorithms was combined into a single code block as shown below. The spectral variant can be just used by passing true value for the 'spectral' parameter for kmeans.

```
data = importdata('Data/Mall_Customers.csv');
data = data.data;
data = data(:,2:5);
num_clusters = [3:10];
SSE = zeros(1, size(num_clusters,2));
for i=1:size(num_clusters,2)
    [cluster_assignments, cluster_centers] = kmeans_cluster(data,
        num_clusters(i));
    SSE(i) = compute_SSE(data, cluster_assignments);
    show_plot(data, cluster_assignments);
    saveas(gcf, strcat('Images/KMeans/Cluster_Arrangement_',int2str(
        num_clusters(i)),'.jpg'));
end
fig = figure; plot(num_clusters, SSE); xlabel('K'); ylabel('SSE'); title('
    Variation of SSE for different values of K');
saveas(gcf, strcat('Images/KMeans/SSE_Convergence.jpg'));


function [cluster_assignments, cluster_centers] = kmeans_cluster(raw_data, k
    , spectral)
%     Code to perform k-means clustering
%     INPUTS:
%           data      =  (n * m) matrix where n is the number of samples and m
%                        is the number of features
%           k         =  integer number of intended clusters
%           spectral =  boolean value representing spectral k-means if true,
%                       else standard k-means
%
%       OUTPUTS:
%           cluster_assignments = labels assigned to the each samples in
%                                 [1,k]
%           cluster_centers     = final clsuter centers found in the process
%                                 of k-means

    % Assigning default values
    if nargin < 3
        if ~exist('spectral')
            spectral=false;
        end
    end

    if spectral
        % for spectral relaxation, map the data samples to k-dimensional
        % feature space
        [U, ~, ~] = svd(raw_data);
        projection = U(:, 1:k);
        rand_mat = rand(k,k);
        orth_mat = orth(rand_mat);
        data = projection * orth_mat;
    else
        data = raw_data;
    end

    [num_samples, ~] = size(data);
    cluster_assignments = zeros(num_samples, 1);
    temp = randperm(num_samples);
    cluster_center_idx = temp(1:k);
    cluster_centers = data(cluster_center_idx, :);
    change = inf;
    count_iter = 0;

    while(change ~= 0)
        % change represents the number cluster assignments that got changed
        % in the current iteration
        count_iter = count_iter+1;
        prev_assignments = cluster_assignments;

        for cur_idx=1:num_samples
            min_dist = inf;
            min_idx = -1;

            % for each sample, find the cluster center which is at min
            % distance
            for cluster_idx = 1:k
                cur_dist = norm(data(cur_idx,:) - cluster_centers(
                   cluster_idx,:));
                if(cur_dist < min_dist)
                    min_dist = cur_dist;
                    min_idx = cluster_idx;
                end
            end
            cluster_assignments(cur_idx,1) = min_idx;
        end

        for cluster_idx = 1:k
            % get the mean of each cluster
            cluster_centers(cluster_idx,:) = mean(data(cluster_assignments
               == cluster_idx,:));
        end

        change = sum(prev_assignments ~= cluster_assignments);
%         fprintf('Number of changes in iter %d: %d\n', count_iter, change);
%         show_plot(raw_data, cluster_assignments);   % plot the clusters
    end

    SSE = compute_SSE(raw_data, cluster_assignments);
    fprintf('Final SSE for k=%d: %f\n', k, SSE);

end


function [SSE] = compute_SSE(data, cluster_assignments)
% Function to compute Sum of Squared Error
% INPUTS:
%   data = the dataset used for clustering
%   cluster_assignments = labels for each sample in the data
%   cluster_centers = the centers found for each cluster
%
% OUTPUT:
%   SSE = final sum of squared errors for the cluster config.

    num_clusters = size(unique(cluster_assignments),1);
    SSE = 0;
    for cluster_no = 1:num_clusters
        cluster_center = mean(data(cluster_assignments == cluster_no,:));
        SSE = SSE + norm(data(cluster_assignments==cluster_no,:)-
            cluster_center)^2;
    end
end
```

```
function [] = show_plot(data, labels)
%   Function to plot the cluster config.
%   INPUTS:
%       data = dataset used for clustering
%       labels = the cluster label assigned to each sample
%
%   OUTPUT:
%       A plot representing the cluster config.

    k = size(unique(labels),1);
    [~, num_features] = size(data);


    if num_features>2
        pcs = pca(data);
        reduced_data = data * pcs(:, 1:2);
    else
        reduced_data = data;
    end

    figure;
    hold on;
    gscatter(reduced_data(:,1),reduced_data(:,2),labels);
    title(strcat('KMeans Clustering Arrangement for K=', int2str(k)));
    hold off;
    pause(2);
end
```

As further experimentation, the mall customer segmentation dataset available at Kaggle: Mall Customer has been used to test the two procedures. It has been observed that the K-means implementation is able to efficiently segregate the data into different clusters. The distribution of the final clusters for both K-Means and Spectral K-Means are for varying values of k are displayed in Figure 1. The corresponding SSE values are further plotted in Figure 2. Although the SSE scores are decreasing as the value of k is increased, it does not mean the clustering is moving towards better labelling. But, it depends on the original number of categories present in the data. So, the value of k plays a crucial role over here. If it is selected appropriately it can produce high quality clustering of the entire dataset. On the other hand, spectral k-means projects the original dataset to a low-dimension feature space. So, it is really hard to visualize the data in the original feature space. Even though the original feature space representation of the cluster arrangement is provided in Figure 1, it is somewhat deceptive because it is not the feature space where clustering happened.

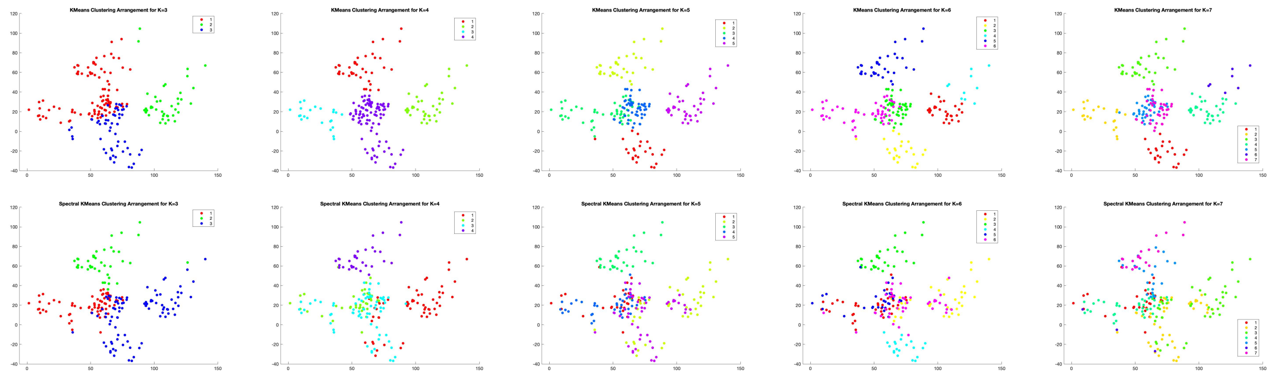
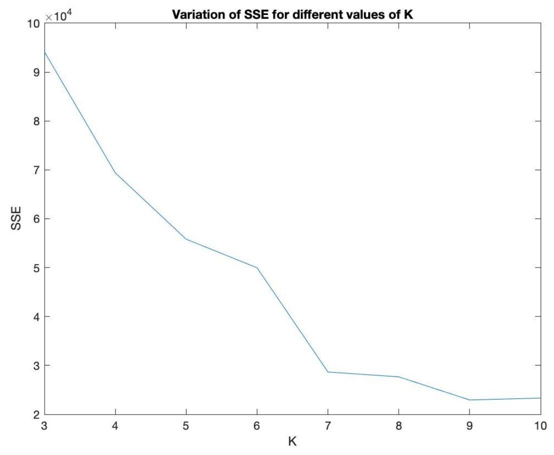


Figure 1: Final Cluster Assignments for KMeans and Spectral KMeans for varying K-values



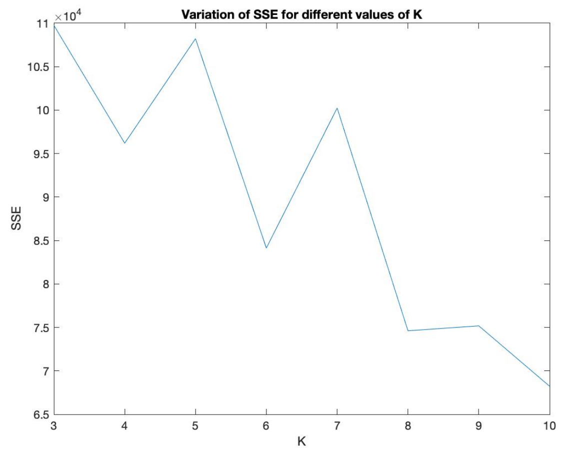
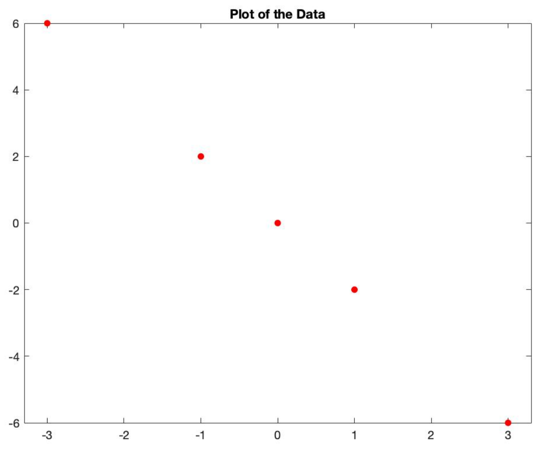


Figure 2: Plot of the SSE scores for different values of k

2 Principle Component Analysis

1. Suppose we have the following data points in 2d space (0, 0), (-1, 2), (-3, 6), (1, -2), (3, -6).
   - Draw them on a 2-d plot, each data point being a dot.
   - What is the first principle component? Given 1-2 sentences justification. You do not need to run Matlab to get the answer.
   - What is the second principle component? Given 1-2 sentences justification. You do not need to run Matlab to get the answer.

Response: The plot of the data in 2d space is shown in the following figure:



From the data, it can be observed that the mean of the data is $(0, 0)$. So, it is already a zero-mean design matrix. The process to find the principal components for the data matrix (let's say X) is to get the right singular vectors for X.

$X = U \times \Sigma \times V^T$ [from SVD]

The principal components of X are just the vectors present in V.

SVD of X gives us the following matrix for V:

$$V = \begin{bmatrix} -0.4472 & 0.8944 \\ 0.8944 & 0.4472 \end{bmatrix}$$

The first principal component of the data is: $[-0.4472, 0.8944]^T$
The second principal component of the data is: $[0.8944, 0.4472]^T$

2. **Experiment**: We apply data pre-processing techniques to a collection of handwritten digit images from the USPS dataset (data in Matlab format: USPS.mat)1. You can load the whole dataset into Matlab by load USPS.mat. The matrix A contains all the images of size 16 by 16. Each of the 3000 rows in A corresponds to the image of one handwrit- ten digit (between 0 and 9). To visualize a particular image, such as the second one, first you need to convert the vector representation of the image to the matrix representation by A2 = reshape(A(2,:), 16, 16), and then use imshow(A2') for visualization. Implement Principal Component Analysis (PCA) using SVD and apply to the data using p = 10,50,100,200 principal components. Reconstruct images using the selected principal components from part 1.

   - Show the source code links for parts 1 and 2 to your github account.
   - The total reconstruction error for p = 10, 50, 100, 200.
   - A subset (the first two) of the reconstructed images for p = 10, 50, 100, 200.

   Note: The USPS dataset is available at USPS Dataset. The image size is 16 by 16, thus the data dimensionality of the original dataset is 256. We used a subset of 3000 images in this homework.

   Response: The steps used to find the principal components using SVD are:

- Center the data by subtracting mean from the samples.
- Perform SVD to get the right singular matrix of the centered data.
- Principal Components are now the columns of the right singular matrix.

The code used for this experimentation is provided below:

```
USPS_data = importdata('Data/USPS.mat');
USPS_mat = USPS_data.A;
p_comb = [10, 50, 100, 200];     % number of principal components considered
num_p = size(p_comb,2);
entry = 1;

% code for simulating PCA reconstruction as many times as user wants
while (entry == 1)
    img_idx = randi(size(USPS_mat,1));
    img_orig = reshape(USPS_mat(img_idx,:), 16, 16);

    figure;
    hold on;
    subplot(1,num_p+1,1); imshow(img_orig'); title('Original Image');
    for i=1:num_p
        recov_data = pca_svd(USPS_mat, p_comb(1,i));
        img_recov = reshape(recov_data(img_idx,:), 16, 16);
        subplot(1,num_p+1,i+1); imshow(img_recov'); title(strcat('p=',
            int2str(p_comb(1,i))));
    end
    hold off;
    saveas(gcf, strcat('Images/PCA_USPS/plot_',int2str(img_idx),'.jpg'));
    pause(3);
    close;

    entry = input('Press 1 for seeing more images, Press 0 to exit: ');
end


function [recov_data] = pca_svd(raw_data, p)
% Function to perform PCA using SVD
%   INPUTS:
%       raw_data = the data used for PCA
%       p = number of principal components to consider
%
%   OUTPUT:
%       recov_data = data recovered from using p principal componenents

    data = raw_data;
    data = data - mean(data);
    [~,~,PC] = svd(data);
    data = data * PC(:,1:p);

    recov_data = data * PC(:, 1:p)';     % reconstructed image
    reconst_error = norm(raw_data - recov_data);
    fprintf('The reconstruction error for p=%d is: %f\n', p ,reconst_error);
end
```

From the experimentation, it was found that the reconstruction error improved as the number of principal components were increased. The exact values are provided in Table 1.

p	**Reconstruction Error**
10	545.720200
50	544.896777
100	544.895231
200	544.894993

Table 1: The Reconstruction Error for different values of p

The most interesting part about this experimentation is the reconstruction of the reduced images. Some examples of reconstructions are provided in Figure 3.

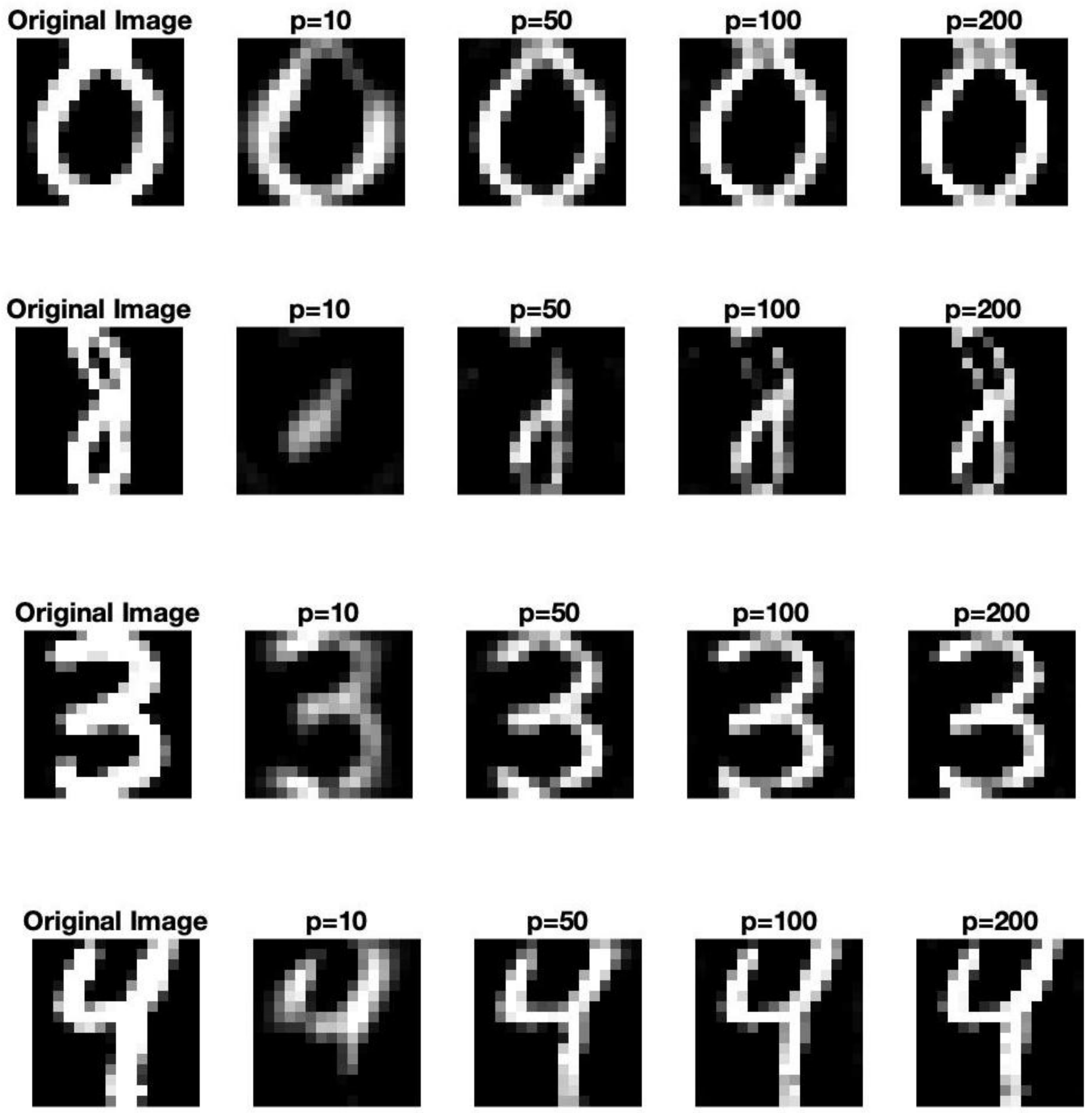


Figure 3: Visual Illustration of the Original and Reconstructed images with different p values