Brenden Hein
Homework 5

CSE 847 (Spring 2022): Machine Learning— Homework 5
Instructor: Jiayu Zhou

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?*

   K-means is an unsupervised learning model that clusters points based on which cluster's centroid they are closest too. However, in order to have accurate clustering, the dataset that uses K-means should be relatively linearly separable. Spectral K-means, on the other hand, instead of trying to construct a matrix of clusters that minimizes the sum of squared error function, using the kth largest eigenvectors of the transposed data matrix multiplied by itself. This can allow for clustering using K-means that may not be linearly separable, adding flexibility to the k-means model. It is possible to arrive at the original K-means using the spectral relaxation for K-means, as well.

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

The code can be found at: https://github.com/Brenhein/Homework-5.git

- *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).*

  Implementing K-means in Python, a random set of K blobs were generated. The user is provided with the ability to enter the number of blobs (the number of clusters in the dataset) and the number of data points. When running, the user is presented with 3 options: Run K-Means, Run Spectral K-Means, and Run Both (which uses the same data set for both K-means and spectral K-Means.

  When running the option that tests both, below is the distribution of the un-clustered data points using K = 5 clusters and N = 500 datapoints

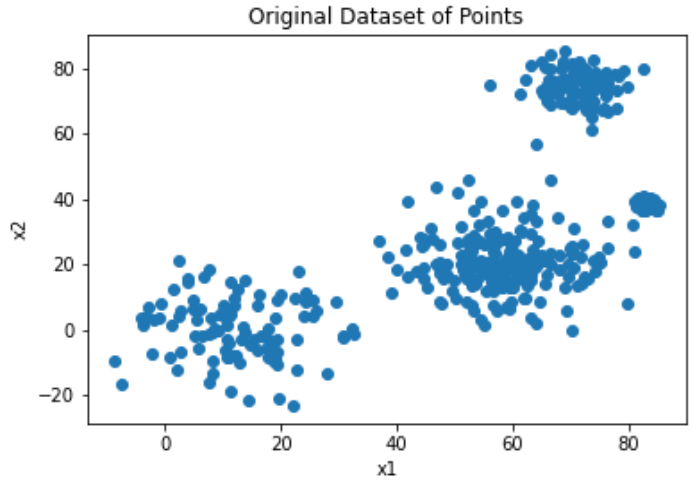

Fig. [1]. The original 500 un-clustered data points plotted

Above the clusters look relatively linearly separable. That makes K-means a great candidate, as it runs on the assumption that the data is linearly separable and formed into blob-like structures. After running 5-means on the 500 datapoints, we get a plot visualized below.

Here we can see that K-means does fairly well. The green and orange clusters at the top-right can be considered one cluster, but as we used a K of 5 clusters, the K-means algorithm decided it would be best to make that cluster 2 separate clusters to fit the number of clusters quota.

Fig. [2]. The 500 data points plotted, with each color representing a cluster and each “x” representing the centroid of said cluster

- *Implement the spectral relaxation of k-means. Create a random dataset and compare the k-means and spectral relaxed k-means.*

  Next, the spectral relaxation of K-means was applied to the dataset. This creates a Gram N x N matrix by multiplying the transpose of X by X. Then the K-largest eigenvectors are discovered from said Gram matrix, and then K-means is used on that eigenvector data matrix instead.

  The results, which are found below, seem incorrect. The issue in question may be related to getting back to plotting the original data matrix based on the clustering results of K-means on the eigenvector data matrix. What was initially attempted was setting each row of the clustered 5-featured eigenvector matrix back to its original 2-featured Euclidean points. Doing this did not seem to work, and after exploring some related research, there was still an inability to generate accurate clustering results, since the spectral relaxation of K-means should work for typical, linearly separable data that K-means would work on.

Fig. [3]. The 500 data points plotted using the spectral relaxation of K-means

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.*

```python
import matplotlib.pyplot as plt


# Plot the dataset
data = [(0, 0), (-1, 2), (-3, 6), (1, -2), (3, -6)]
for point in data:
    plt.plot(point[0], point[1], marker="o", color="blue")

# Save the figure
plt.ylabel("x2")
plt.xlabel("x1")
plt.title("Plot of Points")
plt.savefig("q2a.png")
plt.show()
```

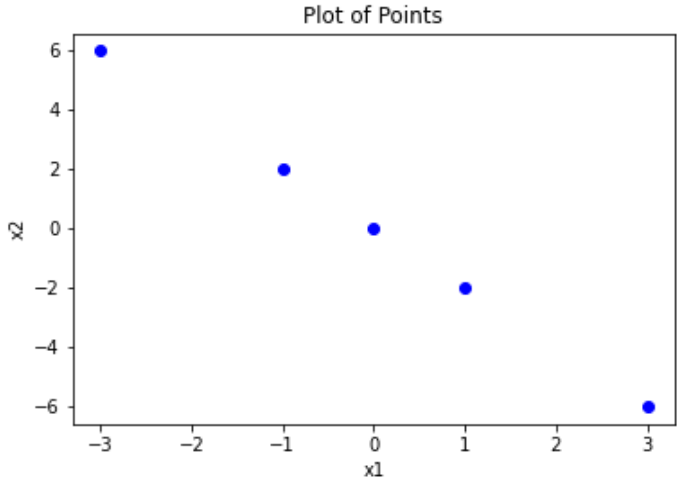

Fig. [4]. A plot of the original datapoints in 2-dimensional space

- *What is the first principle component? Given 1-2 sentences justification. You do not need to run MATLAB to get the answer.*

  The first principal component is in the direction of x2. The reason for this is because the direction of maximum variance appears to be in that of x2, as the spread of x2 is 12, while the spread of x1 is only 6.

- *What is the second principle component? Given 1-2 sentences justification. You do not need to run Matlab to get the answer.*

  The second principal component is in the direction of x1. The reason for this is because the direction of the second maximum variance (since the direction of x2 has already been used) appears to be in that of x1, as the spread of x1 is 6, and it is the only direction left in our dataset.

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 handwritten 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 http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#usps. 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.

The code can be found at: https://github.com/Brenhein/Homework-5.git

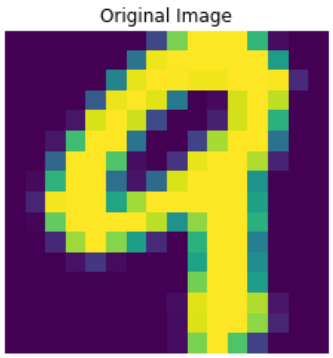

Fig. [5]. The original image plotted on a 16 by 16 grid

[1] https://github.com/jiayuzhou/CSE847/blob/master/data/USPS.mat?raw=true

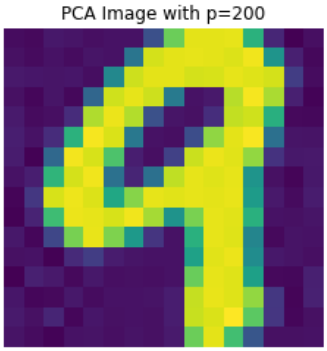

Fig. [6]. The image using the first 200 principal components plotted on a 16 by 16 grid

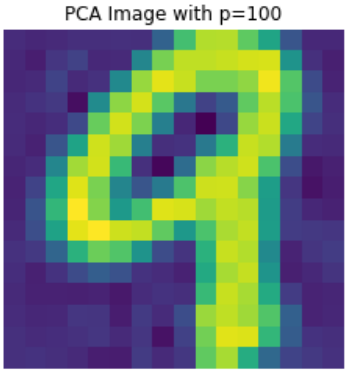

Fig. [7]. The image using the first 100 principal components plotted on a 16 by 16 grid

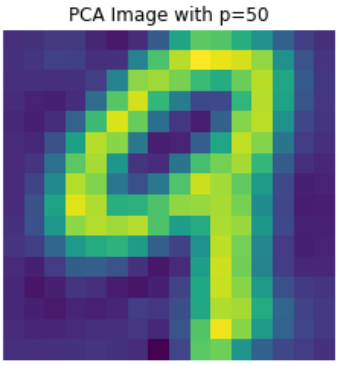

Fig. [8]. The image using the first 50 principal components plotted on a 16 by 16 grid

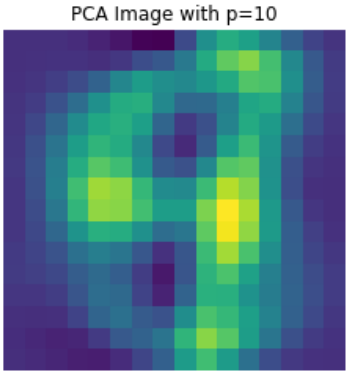

Fig. [9]. The image using the first 10 principal components plotted on a 16 by 16 grid

Here, you can see that by decreasing the number of principal components used, the image appears less and less clear, but even at only 10 principal components, compared to the original 256, the image is still just clear enough to tell that the image is of a “9.” This shows that using dimensionality reduction methods such as PCA will not only reduce the size of the dataset significantly (and help allieviate the curse of dimensionality), but can still classify fairly accurately as its using the most significant principal compenents (which are the ones that explain the vairance of the dataset the best).