Algorithms II - Lab 5

K-Means Clustering

Goals: understand and program K-means

Part I - Clustering Data Sets using K-Means

The first goal of this lab is to write a function which performs K-Means clustering of a finite set of records. Input to your function should include:

- the number of clusters
- the initial centers/generators
- the data to be clustered
- number of attributes of data (this can be determined from the data, if desired)
- number of records in data (this can be determined from the data, if desired)
- tolerance for convergence

The code should output the total cluster variance after convergence is reached plus information for plotting the clusters; the code should output whether or not convergence was obtained. The calling routine for the function should generate several initial choices for centers and choose the result which produces the smallest cluster variance. In the first exercise you will try out your code on a set of data to cluster.

Sometimes when you have data that you want to cluster, you don't know how many clusters to use. In theory, the more clusters we add, the smaller the cluster variance should become. Often however, if we have data which naturally fits into say five clusters, then when we use more than five clusters our total cluster variance will probably not decrease significantly after five clusters. So one way to decide the natural number of clusters is to run the algorithm for different numbers of clusters and plot the cluster variance as a function of the number of clusters. This is explored in Exercise #2.

1. Write a code to implement the K-Means algorithm with the attributes described above using the standard Euclidean distance. As a convergence test use

$$\max_{1\le i\le k} \|c_i^n - c_i^{n-1}\|_2 \le \text{tolerance}$$

where n denotes the iteration number and c_i are the k centers of the clusters.

The data set `bank_notes.txt` contains 200 records containing information about properties of bank notes. There are 100 genuine notes and 100 forgeries. Cluster the data using your K-means algorithm (with two clusters) and compare with the actual clusters via a plot (actual data separated in files `bank_forge.txt, bank_genuine.txt`. For the plot use attributes #4 and # 5. Use a tolerance of 10^{-2}; be sure to run your code with several choices of the initial centers and choose the case which gives the lower cluster variance. When you are done, use the record given below to determine if it is a forgery.

214.9 130.2 130.3 9.2 9.8 140.1

---

2. The data `iris.txt` is a well known data set for clustering; it includes four measurements (sepal length, sepal width, petal length and petal width) based on the petals and sepals of different species of iris; there should be 150 records.

Your task is to use K-Means to cluster this data assuming we don't know apriori how many clusters to use. Let k be the number of clusters. As your initial generators choose k random records from the data file. Calculate the total cluster variance for each value of k. For each value of k you should run several different choices of the initial generators (chosen randomly from the data) and choose the one which has the smallest variance. Plot the total cluster variance versus k for $k = 1, 2, 3, 4, 5, 6$ and choose the value of k which you believe is the natural number of clusters for this data; justify your answer.

Part II - Image Compression using K-Means

If we have a grayscale image then each pixel is represented by an integer between 0 and 255; if we have a color image we know that each pixel is represented by three RGB values creating a myriad of colors. Our strategy now is to choose just a few colors to represent the image. An obvious application of this data compression is when you print an image using a color printer with many fewer colors than are available on your computer. After we choose these colors, then the image chart for the picture must be modified so that each color is replaced by the new color that it is "closest" in color space.

We can use K-Means (or a discrete CVT) to accomplish this image compression. For example, suppose we have a grayscale image and decide that we want to represent it with 32 shades of gray. Our job is to find which 32 colors (out of 256) best represent the image. We then initiate our probabilistic Lloyd's algorithm with 32 generators which are numbers between 0 and 255; we can simply choose the generators randomly. In Lloyd's algorithm we need to sample the space so in our application this means to sample the image; i.e., sample a random pixel. If the image is not too large, then we can simply sample every pixel in the image. We then proceed with the algorithm until convergence is attained. After convergence is achieved we know the best 32 colors to represent our image so our final step is to replace each color in our original matrix representation of the image with the converged centroid of the cluster it is in. For this application we will just use a constant density function and the standard Euclidean distance for our metric.

We will need routines to generate the image chart (i.e., our matrix) from an image and to generate an image from our approximation. There are various ways to do this. One of the simplest approaches is to use the `MATLAB` commands

`imread` - reads an image from a graphics file
`imwrite` - writes an image to a graphics file
`imshow` - displays an image

Specifics of the image processing commands can be found from Matlab's technical documentation such as

`http : //www.mathworks.com/help/toolbox/images/f0 − 18086.html`

or the online help command.

Be warned that when you use the `imread` command the output is a matrix in unsigned integer format, i.e., (`uint8`). You should convert this to double precision (in Matlab `double`) before writing to a file or using it. However, the `imshow` command wants the `uint8` format so if you want to show your final image you must convert back to unsigned integer format using `uint8` command.

---

1. In this problem you will generate a CVT using a probabilistic Lloyd's method and plot it to see that everything is working correctly. Modify your K-Means algorithm to

perform a probabilistic Lloyd's iterative approach to form a CVT for a region which is an n-dimensional box and calculate the cluster variance. Test your code by generating a CVT diagram in the region $(0,2) \times (0,2)$ using 100 generators. Use (i) 10 sampling points per generator, (ii) 100 sampling points per generator, and (iii) 1000 sampling points per generator; for each case display your tessellation using, for example, the `MATLAB` command `voronoi`. Use a maximum number of iterations (300) and a stopping criteria as described above with a tolerance of 0.005. Tabulate the number of iterations required for each case. Plot the cluster variance for each iteration. What conclusions can you draw from your result?

---

2. Use the grayscale image `boat.tiff` and modify your algorithm to obtain approximations to the image using 4, 8, 16, and 32 shades of gray. Display your results along with the original image. As generators choose, e.g., 8 random points between 0 and 255 and because there are a reasonable number of pixels in your image (512^2) you can sample the image by simply choosing each pixel to determine which of the 8 shades of gray it is closest to. Use 10^{-2} as a tolerance in your stopping criteria. After your algorithm has converged don't forget to replace each entry in the image matrix with the center of your cluster and convert to unsigned integer format.

---

3. Use your favorite color image (or `mandrill.tiff`) and modify your algorithm to obtain approximations to the image using 4, 8, 16, 32, and 64 colors. Display your results along with the original image. As generators you will choose, e.g., 8 random points in the RGB color space and as long as there are a reasonable number of pixels in your image (such as 512^2) you can sample the image by simply choosing each pixel to determine which of the 8 colors it is closest to; you can use the standard Euclidean length treating each point as a three-dimensional vector. Use 10^{-2} as a tolerance in your stopping criteria.
