

Lab 8: K-means clustering

Labs on Thursday October 24 and 31, due date on November 6. I suggest to finish the first section of the lab on the 24th!

1. Clustering a Data set 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

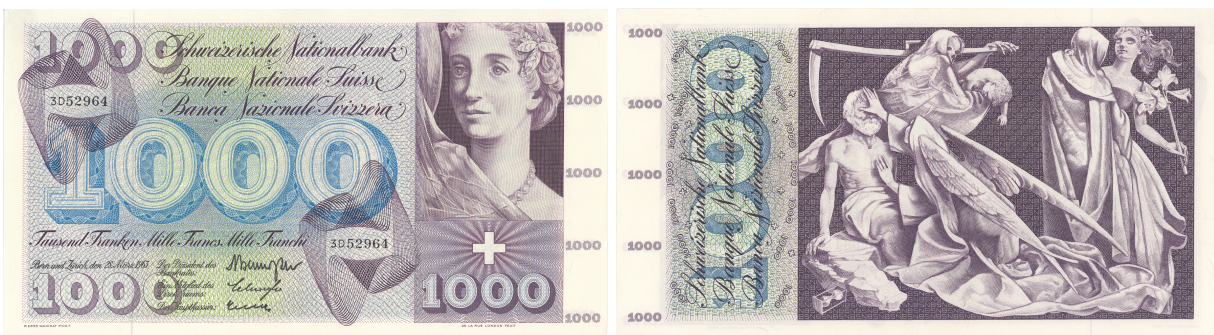
The code should output the total cluster variance after convergence is reached plus information for plotting the clusters; the code should output whether or not convergence was obtained. The calling routine for the function should generate several initial choices for centers and choose the result which produces the smallest cluster variance.

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

$$\max_{1 \leq i \leq k} \|c_n - c_{n-1}\|_2 \leq \text{tolerance}$$

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. here a picture of the front and back of the banknotes (these banknotes were put in circulation in 1957 and were recalled 1980, and became valueless in 2000; old and modern Swiss banknotes are shown here

http://en.wikipedia.org/wiki/Banknotes_of_the_Swiss_franc [the new ones are much more colorful than these]):



Cluster the data using your K-means algorithm (with two clusters) and compare with the actual clusters via a plot (genuine notes are 1 – 100, faked ones 101 – 200). 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. The data consists of 6 measurements of the banknotes, calculate the distance between banknotes as a distance between 6D-coordinates.

Deliverable:

Report a plot comparing the true assignment, and your best k-means assignment, number of mis-assigned notes, and the cluster variance of your plot, the lowest variance reported will get a bonus point.

2. 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 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. If the image is not too large, then we can simply sample every pixel in the image. If the image is large then resort to sample a few pixel (for example a 1000 randomly from the picture and use those to find the the best shades/colors and then map all pixels into the new colors). 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 Matlabs technical documentation such as <http://www.mathworks.com/help/images/basic-import-and-export.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. Use the grayscale image frog_very_small.tif



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 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 (uint8). [I have added two larger pictures of the same frog, that may work with your algorithm].

2. Use your favorite color image (or mandrill.tif)



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 5122) 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.

These files will be available from

<http://people.sc.fsu.edu/~pbeerli/isc4221/clustering>
and

<http://www.peterbeerli.com/classdata/ISC4221/clustering>