Clustering Tutorial
COMS3007

Benjamin Rosman

Instructions: Use your notes and any resources you find online to answer the following questions. You need to submit your answer to QUESTION 3 on Moodle. This can be done in groups of *up to four*. Make sure all your names and student numbers appear on the document!

1. Unsupervised learning algorithms require *unlabelled* training data, but can also be used on labelled data (we just ignore the labels). This is because we are trying to understand something about the structure of the data.

   When developing a model to solve some problem, it is common practice to generate your own training data with known labels. This avoids the need to spend large amounts of time collecting and labelling data, and lets you control exactly how challenging the data set is. This same strategy is useful for studying for machine learning tests and exams!

   An easy way to generate clustering data is through the same process we used for generating classification data: to define a distribution for each cluster. We then also know the ground truth true number of clusters and what they should look like. We commonly use Gaussian distributions. In your favourite language, perform the following tasks:

   (a) Define four 2D Gaussian distributions: with centres at $(1, -1), (-1, 1), (-1, -1), (1, 1)$. Let them both have variances of 1 in each dimension. We'll treat these as the generators of data from four clusters.

   (b) Draw 20 data points from each distribution (use the `randn` function in Matlab or Python). Plot the points from all clusters together. Remember, unlike the classification task, we will be ignoring the class information, i.e., although there are technically four clusters, we will be pretending we don't know this. Now, looking at the data, does it seem obvious that these should be clustered into four groups?

   (c) Repeat the process, with different means and variances. Which datasets do you expect to be easier to cluster?

2. For this question, use the dataset you generated for question 1(b) above. Remember you can discard info on which Gaussian generated each data point, and compress them into a single array. We are now going to cluster this data using k-means.

   (a) Start with something intuitive: let's set $k = 4$, i.e., assume there are 4 clusters in the data. This data is 2D, in x_1 and x_2. Therefore, for each of our k clusters, we need to initialise the cluster centre to be a random point in 2D space. Implement the k-means algorithm from the notes, to iteratively assign points to their closest cluster centre, and then re-compute the cluster centre to be the average of all points assigned to it. Run the algorithm. For each iteration, plot the cluster centres, and colour your data points based on which centre they are assigned to. Do they converge to the expected clusters?

   (b) Make sure you can perform one iteration of the k-means algorithm by hand.

(c) Compute the values of the objective function, the total squared distances between each point and *its own* cluster centre (summed over all points). Compute this for each iteration of the k-means algorithm, and plot the results. What do you notice?

(d) Repeat the above process multiple times with different random initialisations of the cluster centres. Do the final converged cluster centres always look the same? Why?

(e) Now run the k-means algorithm with 2 clusters instead of 4. What do you notice? Now repeat this for $k = \{3, 5, 6, 7, 8\}$ clusters.

(f) For each of the different values of k above, compute the error (objective function value), and plot these values. What do you notice? From this plot, what is the best value of k? Why?

3. We are now going to perform image colour segmentation / image compression. Load an image into your editor as a 3D matrix (the third dimension should have three values, for the red, green and blue values of each pixel). You can do this using `imread()` in Matlab or Python (in matplotlib in Python). You can use any image for this, or try with the one on Moodle: `peppers.bmp`. You can draw the image using the `imshow()` command.

   Submit your code for this question, as well as TWO images (one could be the peppers image, but anything is fine) with the images redrawn using 4 clusters.

   (a) Load the image. For clustering, we don't care about the x and y positions of each pixel: we just want to group all the colours into k clusters. The current dimension of your matrix is $x \times y \times 3$. Reshape this into a matrix that is $N \times 3$, where $N = x * y$ is the number of pixels.

   (b) Choose $k = 2$. Now run the k-means algorithm on this data as you did in the previous question. When this converges, you should have two cluster centres, and every colour is assigned to one of them. Each cluster centre is a point in 3D colour space, i.e., $c_1 = (r_1, g_1, b_1)$ and $c_2 = (r_2, g_2, b_2)$, which will be the average of every colour assigned to it.

   (c) Now redraw the image. You need to take the original $x \times y \times 3$, but for each (x, y) pixel, replace its three values with the values of the appropriate cluster centre. Draw the final image using the `imshow()` command. If you did everything correctly, it should still resemble the original image, but only have two colours.

   (d) Repeat this process for $k = \{4, 8, 16, 32, 64\}$. What do you notice?