COM2004/3004 Lab Week 10
k-means clustering

Objective

In the lab you will use hard k-means clustering to cluster a set of 2D points. You will see that the result is highly dependent on the initial cluster centre estimates.

Background

In an earlier lab class you experimented with agglomerative clustering. In this class you will be using a function optimization approach called *hard k-means clustering*. An efficient implementation has been provided for you.

1. Download the data and tools

Make a folder for the lab class and from MOLE download the following files.

- `kmeans.m` – code to perform the clustering.
- `makeClustersEqual.m` – makes clustered data in which clusters all have the same spread.
- `makeClusters.m` – as above except the spread of each cluster can be very different.

2. Generate and display some data to be clustered

Start MATLAB and use the program `makeClustersEqual` to make some data *x*. We will make data that falls into 2 clusters each containing 100 points.

```
npoints = 100;
nclusters = 2;
x = makeClustersEqual(nclusters,npoints);
```

Display the data using a scatter plot

```
scatter(x(:,1), x(:,2));
```

3. Running the k-means algorithm

Load the code `kmeans.m` into an editor and study it carefully. Check it against your lecture notes and **make sure that you understand how it is working**. It has been written to be efficient so you might have to look at it quite carefully to really understand what each line is doing. Ask a demonstrator for help if it is not clear. There are a few comments in the code that should help.

The *k*-means algorithm can be initialized by randomly assigning positions to the cluster centres. We are going to set the *x* and *y* coordinates of each cluster centre to be random numbers between 0 and 100.

```
w = rand(2, nclusters) * 100;
```

Now we are ready to perform the clustering. Run the clusterer by typing,

```
[w2, bel] = kmeans(x', w, true);
```

Take care to use the transpose of the x matrix, (i.e. `x'`). This is needed simply because the clusterer expects the points to appear as column-vectors whereas `makeClustersEqual` generates them as row vectors.

The last parameter, 'true', is a switch that tells `kmeans` to produce an animated scatter plot while performing the clustering. If you don't want to watch the clustering being performed then either set this to 'false' or simply omit the parameter (the code runs faster when plotting is turned off).

Look at `w2`. This will show you the final estimate of the cluster centres stored as a pair of column vectors. The index of the cluster to which each point belongs is stored in the vector `bel`.

If you have run `kmeans` with the plotting option turned off you will need to display the output clusters by using a scatter plot and distinguishing clusters by using either red or blue points. We can select all the points belonging to cluster 1 by using `x(bel==1)` and cluster 2 with `x(bel==2)`. So we can colour code the scatter plot by typing,

```
figure;
hold on;
scatter(x(bel==1,1), x(bel==1,2), 'r');
scatter(x(bel==2,1), x(bel==2,2), 'b');
```

4. Experimenting with different initializations

Save all the commands from **section 3** into a script file called `recluster.m`.

Now rerun `recluster.m` with the same data and compare the final clusterings. Are they always the same? Run the code several times over and over and observe what happens. You should see that sometime the clustering works well and other times it produces unexpected results.

5. Experimenting with harder problems

Now generate data with a larger number of clusters, say four. Run the clustering code multiple times and keep a note of how often it produces a result that looks correct.

Try generating data using `makeClusters` rather than `makeClustersEqual`. This code can produce clusters that have very different spreads. Does the *k*-means clusterer perform well on this data?

You should find that although *k*-means is very efficient, it only works well in certain situations, and it is quite dependent on the initial cluster position estimates. Soft *k*-means can give better results. Better still is to use a Gaussian mixture model and estimate the parameters using the EM algorithm (lecture on Friday). However, none of these optimization techniques is immune to the problem of poor initialization. If the initial cluster centres are too far from the correct values the algorithms are prone to getting stuck in local optima. These local optima may be very different from the overall best clustering.