COM2004/3004 Lab Week 9
Agglomerative Clustering Challenge

Objective

In this lab class you will gain some practical experience with hierarchical agglomerative clustering. You will start by playing with some 2D data and seeing how different distance measures can produce different clustering results. You will then apply clustering techniques to the word search letter data. You will be challenged to find a 10-dimensional representation of the letters that allows the data to be reliably clustered. There will be a prize for the best result.

Background

In the lectures we have been discussing clustering. We have seen that clustering lies at the heart of many data driven applications. In particular, it can be used to solve problems in situations where we do not have access to labeled training data. For example, the word search problem (assignment stage 2) could be solved entirely without labeled data: We could take our training letter images and simply cluster them into 26 groups. We would hope that each letter would fall into a separate cluster. Each cluster could then be assigned an arbitrary label (e.g. integers 1 to 26). We would then take an image of a word search grid and of the words to be found. We would classify the letters in the grid and the letters in the search words using the arbitrary labels (1 to 26). Finally we would search for the words by matching the label sequences of the search words to the labels in the grid.

For the above 'unsupervised' word search solving approach to work, it would be necessary for the initial clustering step to be accurate, i.e. each cluster should only contain examples of one letter. In today's practical we are going to see if an accurate clustering can be achieved using a simple *agglomerative hierarchical* clustering approach. (The algorithm will be explained in detail in the lecture on Friday).

1. Download the data and tools

Make a folder for the lab class and download the following files from Week 9 on MOLE,

`lab9_data.mat` – the letter data.

`makeClusters.m` – for making random clustered 2D data.

`cluster.m` – some flexible agglomerative clustering code.

`doPCA.m` – code for applying PCA transform to data.

2. Loading the lab data

Start MATLAB and load the `lab9_data` (i.e., "`load lab9_data.mat`").

The following familiar variables should appear in your MATLAB workspace,

```
test_data,  test_labels
```

As in previous weeks, the matrix `test_data` contains a set of feature vectors arranged as rows. Each feature vector contains the 900 pixel values that when arranged as a 30 by 30 matrix form the image of a single character. The character labels have been provided and are stored in the variable `test_labels`. The labels will not be needed by the clustering algorithm (remember, clustering is unsupervised), however, it is useful to have access to the labels so that we can eventually evaluate how well the clustering algorithm has performed.

Remember, you can view the *ith* character in the data set by typing,

```
imagesc(reshape(test_data(i,:),30,30));

colormap(gray);
```

3. Clustering 2D data

We will start by testing the clustering code on some 2D data. We can plot 2D data using scatter plots so it is easy to see exactly what is going on. A MATLAB program, `makeClusters.m`, has been provided that can generate a suitable set of 2D points that fall into clusters. The makeClusters program takes two parameters,

```
x = makeClusters(number_of_clusters, points_per_cluster);
```

It will generate 2D points stored in a matrix `x` with a separate row for each point. For example, to make 10 clusters with 10 points in each cluster type

```
x = makeClusters(10,10);
```

We can now display the generated points using a scatter plot as follows,

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

Each time you run `makeClusters` you will get a different result, so if the clusters are not nicely spread out run it a few more times until they are.

We will now use the clustering program to cluster the data in `x`. Type (or cut and paste) the following command **very** carefully,

```
cluster(x,10,'@(x,y)dm_cc_min(x,y,@dm_pp_euclid)','@display_points');
```

You should see an animation of the points clustering with colours being used to indicate which points belong to which clusters. Does the final clustering seem reasonable?

Try generating data that has fewer or more clusters. What happens if you tell the clusterer to find fewer clusters than are actually present?

The `makeClusters` has an optional 3rd parameter that sets the spread of the points within each cluster. It is by default set to 20. Try making clusters that overlap by setting the spread higher. For example, the following command should make a pair of clusters that overlap somewhat,

```
x = makeClusters(2, 40, 100);
```

Run the clusterer on the overlapping clusters. How does it perform?

4. Experimenting with different distance metrics

The cluster program has the following parameters

```
cluster(data, N, algorithm_string, display_string);
```

`data` – the feature vectors to be clustered stored as rows of a matrix.

`N` – the desired number of clusters to find.

`distance_string` – a string describing the distance measure.

`display_string` – a string describing the type of animation display.

In Section 3 we ran the cluster program with `distance_string` set to `'@(x,y)dm_cc_min(x,y,@dm_pp_euclid)'`. With this setting the clusterer will use a minimum distance cluster-to-cluster measure where point-to-point dissimilarities are computed using a Euclidean distance metric. Other strings that you can use are shown below. (With help from the lecture notes you should be able to guess what they mean).

```
'@(x,y)dm_cc_min(x,y,@dm_pp_euclid)'

'@(x,y)dm_cc_max(x,y,@dm_pp_euclid)'

'@(x,y)dm_cc_mean(x,y,@dm_pp_euclid)'

'@(x,y)dm_cc_average(x,y,@dm_pp_euclid)'


'@(x,y)dm_cc_min(x,y,@dm_pp_manhattan)'

'@(x,y)dm_cc_max(x,y,@dm_pp_manhattan)'

'@(x,y)dm_cc_mean(x,y,@dm_pp_manhattan)'

'@(x,y)dm_cc_average(x,y,@dm_pp_manhattan)'
```

```
'@(x,y)dm_cc_min(x,y,@dm_pp_negative_cosine)'

'@(x,y)dm_cc_max(x,y,@dm_pp_negative_cosine)'

'@(x,y)dm_cc_mean(x,y,@dm_pp_negative_cosine)'

'@(x,y)dm_cc_average(x,y,@dm_pp_negative_cosine)'
```

(Note 'cosine distance' is a *similarity measure*. This particular clustering code expects the proximity to be measured in terms of *dissimilarity*. So it is actually *negative cosine distance* that is used).

Try reclustering the same 2D data using different distance measures.

5. Clustering the letter data.

We are now going to try and cluster the 900 dimensional letter data. We will set the desired number of clusters to be 26 because we know there are 26 different letters in the alphabet. I have provided a special display option, `'@display_images'`, that will show the letters as images rather than as 2D points.

The test data has 200 letters but we will only use the first 100. (If you want to use more you will have to be very patient!) To cluster the first 100 letters type the following command **very carefully**,

```
c=cluster(test_data(1:100,:), 26, '@(x,y)dm_cc_mean(x,y,@dm_pp_negative_cosine)','@display_images');
```

Once the code starts running you will see that clusters are displayed as separate columns of letter images. Letters will start out in 100 separate columns and by the end of the process there will be just 26 columns remaining. (You may need to stretch the Figure window to be short and wide to make the letter tiles square enough to read clearly).

Are the letters clustering nicely?

6. Clustering with Principal Component Analysis features

We will now try clustering the letters in a lower dimensional space. We will use PCA to perform the dimensionality reduction. Let's reduce the dimensionality from 900 down to just 10.

We have seen how to do this in the last lab class, but I have provided some handy code for you called `doPCA.m`. To use it type,

```
pca_data = doPCA(test_data(1:100, :) , N);
```

where `N` is the desired number of PCA features, e.g. 10.

Then type,

```
c=cluster(pca_data, 26, '@(x,y)dm_cc_mean(x,y,@dm_pp_euclid)', '@display_points');
```

JPB, 20/11/2012

Remember, the first PCA coefficient is often not very discriminative. So we might get a better results doing

c=cluster(pca_data(:,2:10), 26, '@(x,y)dm_cc_mean(x,y,@dm_pp_euclid)', '@display_points');

Note, we are using 'display_points' (display_images was only written to work with 900 element image vectors). However, there is a problem when trying to visualize clusters in more than two dimensions. display_points can only show 2D points so it only uses the first two components of each feature vector. However, some points that are far apart in 10-D space can look quite close when seen in 2-D so the clustering might not seem sensible when seen in the 2-D plane. (Consider, by analogy, stars that are grouped into the same constellation that can often be hundreds of light years apart and only appear to be close together when seen on the 2D sky. This `apparent closeness of distant points' effect will occur far more often when projecting from a 10-D space onto a 2-D plane).

7. Evaluating the clustering algorithm

We need to find an objective way to evaluate the clusterings. This is possible because we happen to have access to the correct labels. (In many problems the 'correct' label may be unknown or unknowable).

The cluster code returns a cell array that stores which points are in each cluster. We have stored this result in the variable c.

Type c{1} and MATLAB will display a vector of indexes of test_data examples that have been put in the first cluster. Type c{2} and you will see the second cluster, etc. To see the labels of the members of each cluster we can do,

```
test_labels(c{1})

test_labels(c{2})
```

and so forth. If the clustering were perfect all the labels in one particular cluster would be the same.

Think how you could write a bit of code that would take the cell array, c, and the label data, test_labels, and return a score telling you how many letters are in 'the wrong' cluster. This seems straightforward at first sight, but after some thinking you'll see it is a difficult problem in itself. (I'll provide some code after you've had a chance to think about it. You will need this evaluation code for testing the performance of the features you submit for the challenge described below.)

PTO

CHALLENGE: Deadline Sunday 1st December

I will be awarding another prize to the person who can find a 10 dimensional representation of the letter data and a distance metric that together provide the best agglomerative clustering.

To enter the challenge you need to submit code that will take a matrix of 100x900 pixels representing 100 letters and will return a matrix of 100x10 values representing the dimensionally reduced data. You must also state the `distance_string` that you want the cluster program to use.

To avoid excessive 'overfitting' I will be evaluating the features on a secret random set of 100 examples from the training data. But feel free to test your code on as many different 100 sample subsets as your wish in order to tune your solution.

There will be a prize for the winning entry awarded during the Friday 5th December lecture.

JPB, 20/11/2012