Aston University
Machine Learning

Unsupervised Learning (2)

Learning Outcomes:
In the current unit, we continued to study clustering algorithms. We learned about two further clustering paradigms - density-based and hierarchical – and about some methods for cluster evaluation. In this lab, we will put our knowledge into practice using the Python library scikit-learn.

Note that you may have implemented two of the algorithms dealt with in this lab (k-means and the EM algorithm for fitting a GMM) in the previous lab. The aim of implementing the algorithms from scratch was to solidify your understanding of them. In practice, when using standard algorithms, it is better to use an implementation in a well-known library (such as scikit-learn) as such libraries are well tested and likely provide associated utility functions to assist with tasks such as cluster evaluation.

Instructions:
We will be using the same datasets in this lab as we used for the previous one. Go back to your work for the previous lab and look again at the scatter plots you generated of the four datasets to remind yourself of the distribution of the data.

Task 1:
In this task, you will be using scikit-learn to apply the k-means algorithm to each of the datasets and to decide on a reasonable number of clusters.

For each of the four datasets use scikit-learn's implementation of k-means (documentation here) to the data into 2 clusters. Plot a graph of the resulting clusters. Check that the output is similar to what you observed with your own implementation.

Now, use the "elbow" method discussed in lectures to determine a reasonable number of clusters for each of the datasets. You will need to:

- Apply k-means to the dataset for a range of values of k (for this example, try k values ranging from 2 to 10).
- For each clustering, calculate some appropriate measure of error. For this example, you can use the `inertia_` attribute of a computed clustering which can be calculated as follows:

```
>>> from sklearn.cluster import KMeans
>>> kmeans = KMeans(n_clusters=2).fit(X) #for some dataset X and 2 clusters
>>> kmeans.inertia_
```

The `inertia_` attribute gives you the sum of squared errors, which is equal to the reconstruction error discussed in lectures multiplied by the number of datapoints. It can be used in the same way as the reconstruction error to evaluate a clustering.

- Store the error for each value of k. Create a plot of k against error. Use your knowledge from lectures to decide what a good value for k would be.

As an extension, you may want to experiment with silhouette analysis to choose a good value for k (example here). Are your values of k chosen using the two different methods similar?

Task 2:
In this task, you will be using scikit-learn to use the EM algorithm to fit a GMM to each of the datasets and to decide on a reasonable number of clusters.

Your approach to this task should be the same as in the previous one, with the following differences:

- You will need to fit a GMM to the data (documentation here).
- As GMMs are probabilistic models, you can base your evaluation on a model's likelihood (or log likelihood). This can be calculated directly using the `score` method in scikit-learn. However, I recommend using the Bayesian Information Criterion (the `bic` method) – a modification to the log likelihood which helps prevent overfitting.

You can find an example of using the Bayesian Information Criterion for model selection in the context of GMMs here.

Task 3:
In this task, you will be using scikit-learn to experiment with the new algorithms covered in this unit: DBSCAN and agglomerative clustering.

You should now understand scikit-learn well enough to be able to use its documentation which, for the algorithms in question, can be found here:

- DBSCAN
- Agglomerative clustering

Note that the agglomerative clustering implementation requires you to specify the number of clusters. The full dendrogram can still be explored using the `children_` attribute. You can also use the `distance_threshold` parameter to specify a value at which to "cut" the dendrogram instead of specifying a number of clusters.

Apply the algorithms to the datasets and visualise the results.