CASC345B/CSCM45J Lab Class 2 **Release Date: 16/02/2024**
Total Marks: 4 **Due Date: 01/03/2024**

This lab is about utilizing unsupervised learning to cluster data from the Fisher Iris dataset. We will be implementing the k-means and GMM clustering algorithms on some example data by adding our own code to a Python notebook. Packages used in this lab are: `numpy`, `matplotlib`, and `scikit-learn`.

Create a new Notebook, titled `<student_number>_Clustering.ipynb` to complete this labtask. Again, use markdown to annotate your notebook as required.

☐ Task 2.1 – Fisher Iris Dataset

The first task is to get used to the provided dataset and explore the observed features. Both the data and labels are contained within numpy arrays in the available files on Canvas.

- Download `Iris_data.npy` and `Iris_labels.npy` from Canvas, and place them in your working directory.
- Using a Markdown cell, provide annotation describing the dataset. How many samples? What features? What does `Iris_labels.npy` contain?
- Load both `Iris_data.npy` and `Iris_labels.npy`, and visualise `Iris_data` with a scatter plot similar to last week. This time however, use `Iris_labels` to color the scatter plot (hint: check the API of matplotlib's `scatter` function). Remember we can only plot two of the features against each other for the moment. Don't forget to label your axes and give the plot a title.

At the end of task 2.1 you should have something similar to Figure 1, depending on the feature dimensions you choose to visualise:

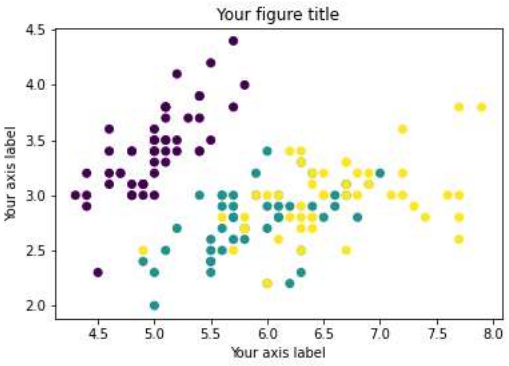

Figure 1: Fisher Iris dataset.

Note: You can also apply a legend to the figure and provide further detail, if you know the Iris species. Check out the matplotlib.pyplot API for more functionality.

☐ Task 2.2 – k-Means

The second task is to cluster our Iris data with k-means, using unsupervised clustering to attempt to identify groupings within the data in a data-driven (bottom-up) way. You must use **all 4 of the feature dimensions** within the Iris dataset in order to cluster the observations we have into k clusters. I leave it to you to select a suitable value for k.

This task will build a model from the scikit-learn package, and I encourage you to check the documentation available online. The latest API can be found at `https://scikit-learn.org`; however this can be tricky to navigate, so a search for "sklearn <thing of interest>" in your favourite search engine will suffice. It is important to check the API, as it describes expected inputs and outputs, and can include parameters not discussed in the lab.

The main model of this task is implemented via the `KMeans` class in sklearn; the API can be found at `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans`. Please read through this page, as it includes the constructor signature, a description of parameters and attributes, and also includes the methods available to the class (such as `.fit()` and `.predict()`). This API will help you when answering the following tasks.

- Create a new Markdown cell explaining the code that is to follow.
- Initialize an instance of a scikit-learn `KMeans` object. To do so, you will need to import `KMeans` from `sklearn.cluster`. You might want to consider what the constructor's default parameters are.
- Fit your k-means model to the `Iris_data`, using all 4 feature dimensions.
- Predict the cluster membership of the samples within `Iris_data`.
- Produce a scatter plot from the results of the prediction, and compare this to your plot of the true labels from task 2.1. Note that our cluster IDs are unsupervised, so will not necessarily map to the same IDs within `Iris_labels`.
- Add the k cluster centroids of the model to your plot (hint: check the attributes of a KMeans object in the API).
- Consider the API and what arguments you could use to produce a better clustering model. Try creating a new KMeans object. What do the `init` and `n_init` parameters do?

At the end of task 2.2 you should have something similar to Figure 2, depending on the feature dimensions you choose to visualise and your model's trained state:

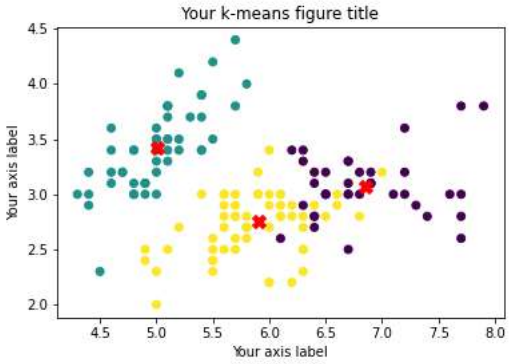

Figure 2: k-means clustering applied to Fisher Iris dataset. Red X's denote clusters.

☐ Task 2.3 – Gaussian Mixture Models

The third task is to cluster our Fisher Iris data with a GMM. For this task, we will implement the GMM algorithm in our notebook using the `GaussianMixture` class of scikit-learn, clustering our data with the posterior probabilities of g Gaussian distributions. We will again use all four of the available feature dimensions.

The main model of this task is implemented via the `GaussianMixture` class in sklearn; found at https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture. Again, take a look at the API and get familiar with it, it will help you in the task.

- Create a new Markdown cell explaining the code that is to follow.
- Create a code cell, in which you initialize an instance of a scikit-learn `GaussianMixture` class. To do so, you will need to import `GaussianMixture` from `sklearn.mixture`.
- Fit the GMM model to the `Iris_data`.
- Predict the cluster membership of the samples within `Iris_data`.
- Produce a scatter plot from the results of the prediction and compare them to both the true labels from task 2.1 and the k-means predictions from task 2.2. Again our predicted cluster IDs are unsupervised, so will not necessarily map directly.
- Add the g gaussian means of the model to your plot using `plt.scatter()`.

At this point in task 2.3, you should have something similar to Figure 3, depending on the feature dimensions you choose to visualise and your model's trained state:

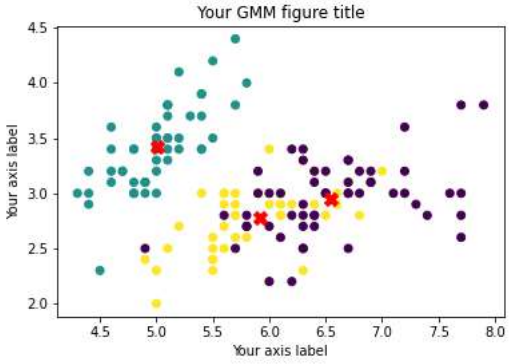

Figure 3: GMM clustering applied to Fisher Iris dataset. Red X's denote clusters.

Note from the lectures about how k-means and GMMs differ, and their usage of hard and soft boundaries. In this GMM task we would like to see the posterior probability of each data point belonging to a given Gaussian component, as this shows us the soft nature of the GMM clustering.

- Predict the posterior probabilities for the data points in the `Iris_data`. (hint: look at the API for the `.predict_proba()` function, and check the output's shape and range).
- Produce a scatter plot (or subplot) **for each component/Guassian** of the GMM model. Within each scatter plot, plot the component's posterior probabilities as the color intensity for each sample. This should result in g different plots which have a continuous value rather than cluster IDs. Provide appropriate labels, titles, and colorbars to each plot.

- Go back and tweak your `GaussianMixture` model. Consider the API and what arguments you can use to produce a better clustering model. Re-run your cells to see if it improves. What does the `init_params` parameter to the constructor do, and how does it relate to the slides on GMMs?

At the end of task 2.3 you should have something similar to Figure 4, depending on the feature dimensions you choose to visualise and your model's trained state:

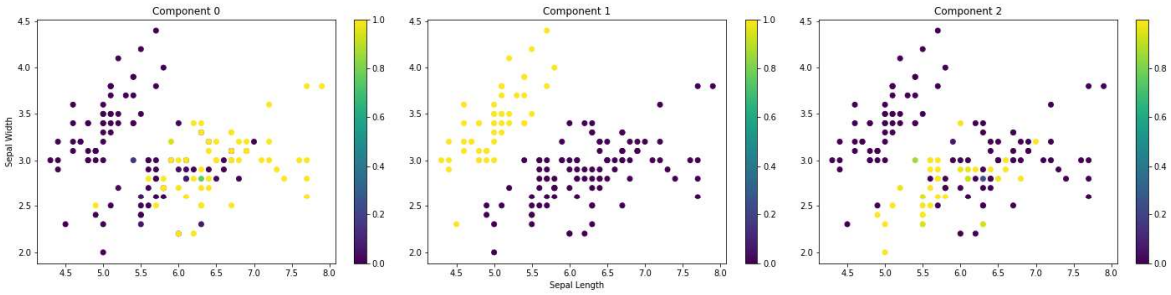

Figure 4: Visualisation of each Gaussian component's posterior distribution.

☐ Task 2.4 – Apply your skill.

On Canvas there is the numpy datafile `task2_4.npy`. Download this file to your current working directory. Using the skills from the previous 3 tasks, cluster the data and see if you can find the true number of underlying clusters. Include visualisations and markdown to provide some reasoning for your work. Hint: the true number of clusters is not two.

The task here is not to get the right answer (although well done if you can), but rather to show your working.

☐ Challenge Task 2.5

K-Means utilises the following equation to iteratively update the cluster membership of the training data, updating the position of the k cluster centroids through a repetition of assignment and update steps. The following describes the function to be minimised:

$$\sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} \text{dist}(\mathbf{x}, \mathbf{m}_j)^2$$

GMMs utilise the following equations to achieve their clustering, this time by iteratively repeating Expectation and Maximisation steps. The following describe the update of the GMM parameters for the mixing coefficient, the Gaussian μs, and the posterior probability covariance matrix:

$$P(j)^{new} = \frac{1}{N} \sum_{n}^{N} p^{old}(j|x^n) \tag{1}$$

$$\mu_j^{new} = \frac{\sum_n^N p^{old}(j|x^n) x^n}{\sum_n^N p^{old}(j|x^n)} \tag{2}$$

$$\sum\nolimits_j^{new} = \frac{\sum_n^N p^{old}(j|x^n)(x^n - \mu_j^{new})(x^n - \mu_j^{new})^T}{\sum_n^N p^{old}(j|x^n)} \tag{3}$$

On Canvas you will find the jupyter notebook `ClusteringFromScratch.ipynb`. Download this file and have a look inside. Correlate this with your notes from the lectures. Identify where in the code these equations are being implemented.

Consider why it appears the GMM implemented from scratch performs worse than the GaussianMixture model from scikit-learn. Hint: Check out the API and the `init_params` argument. How could we change our implementation to match the default in scikit-learn's `GaussianMixture`?