CS-345/M45 Lab Class 2 — Release date: 10am 12/10/2020, Due date: 10am 26/10/2019

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 Jupyter notebook. Packages used in this lab are: `numpy`, `matplotlib`, and `scikit-learn`.

Create a new Jupyter Notebook, titled `ClusteringNotebook.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 our working directory.
- Load the data using numpy, storing them in appropriate variables.
- Use Markdown above where you load the data, providing annotation describing the dataset. How many samples? What features? What does `Iris_labels.npy` contain?
- Visualise `Iris_data` with a matplotlib scatter plot, using `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 eachother for the moment. Don't forget to label your axes and give the plot a title.

☐ Task 2.2 – k-Means

The second task is to cluster our data with k-means, using unsupervised clustering to attempt to identify groupings within the data in a data-driven way. We will use **all of the feature dimensions** within the Iris dataset in order to cluster the observations we have into k clusters.

- Create a new Markdown cell explaining the code that is to follow.
- Initialize an instance of a scikit-learn `KMeans` class. To do so, you will need to import `KMeans` from `sklearn.cluster`.
- Fit the model to `Iris_data` (hint: check the API for the `fit()` function).
- Predict the cluster membership of the samples within `Iris_data` (hint: check the API for the `predict()` function).
- Produce a scatter plot from the results of the prediction. Note here that our predicted 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 within a KMeans object and use `plt.scatter()`).
- Go back and tweak your `KMeans` model. Consider the API and what arguments you could use to produce a better clustering model. Re-run your cells to see if it improves.

□ 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 of the available features.

- 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 model to `Iris_data` (hint: check the API for the `fit()` function).
- Predict the cluster membership of the samples within `Iris_data` (hint: check the API for the `predict()` function).
- Produce a scatter plot from the results of the prediction. Note here that our predicted cluster IDs are unsupervised, so will not necessarily map to the labels within `Iris_labels`.
- Add the g gaussian means of the model to your plot (hint: check the attributes within a `GaussianMixture` object and use `plt.scatter()`).
- Predict the posterior probability of each component (Gaussian) given the observed data within `Iris_data` (hint: check the API for the `predict_proba()` function).
- Produce a scatter plot **for each component** 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. Provide appropriate labels and titles for your plots. Add a colorbar 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.

□ Challenge Task 2.4

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:

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

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 Gaussian-Mixture 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`?