Homework Problem Set 6: Ensemble methods, imbalanced classes

- Due Wednesday, Sunday, April 28 at 11:59 PM

- Upload a pdf to Canvas

  Each question is worth the same number of points.

- You are allowed to use ChatGPT or other technologies. However, you are REQUIRED to provide the prompts and responses, in addition to the code you submit. Specify which chatbot you are using and its version. Failure to do so will penalize your score.

- *All* your plots have title, x and y axis labels, and with a grid. They are to be returned in the `answers` Python dictionary and put them in a report uploaded to Canvas. The programming portion counts for 100 points. The report counts for 30 points. The title of your plots should include the case run and the parameters used. If you use the enter symbol "
  n" in your titles, they can be multiline. In all plots, adding the command

```
plt.tight_layout()
```

  before `plt.savefig` will beautify your plot.

- Save all your figures in pdf or png format. pdf gives maximum quality which is not affected by scaling. png format has no compression. Avoid jpeg files.

- In your report, add explanatory notes with your figures. All homeworks are individual. Copying that is identified leads to zero on the assignment.

Question 1: (50pts)
You are given two files: `cluster_data.npy` and `cluster_labels.npy` (read up on how to read these files). "cluster_data.npy" contains an array of shape $50,000 \times 2$, the points that you are to cluster. You will use 10,000 at a time to create your clusters. Conduct hyperparameter studies on the first 10,000 points (data[0:10000,:], labels[0:10000]) for three different clusteirng algorithms (see below) and then use these hyperparameters on five slices of data `data[i*10000 : (i+1)*10000]`, `labels[i*10000 : (i+1)*10000]`, $i = 0, 1, 2, 3, 4$. The cluster data and labels is stored in file `question1_cluster_data.npy`, `question1_cluster_labels`. You are only allowed to use the `numpy` module for this question. Plot the data read in with a scatterplot (`plt.scatter`).

Develop and run three clustering algorithms as follows:

DENCLUE algorithm

- Write a function to run the DENCLUE algorithm. Choose appropriate parameters. Return the parameters used in a dictionary. You are to implement the algorithm using only the `numpy` module.

- Run a spectral clustering algorithm with five clusters. Choose appropriate parameters. The hyperparameter study uses only the σ and ξ parameters. σ is the parameter used in the Gaussian Kernel function

$$K(x,y) = \exp\left(-\frac{\text{dist}(x,y)}{\sigma^2}\right),$$

where `dist(x,y)` is the Euclidean distance between points x and y. ξ is a threshold paraemters. Points with an amplitude below the threshold ξ are not part of a cluster. Return the parameters used in a dictionary. Implement the algorithm using only the `numpy` module.

Spectral algorithm

(a) Write a function to run the spectral clustering algorithm. Choose appropriate parameters. Return the parameters used in a dictionary. You are to implement the algorithm using only the `numpy` module.

- N_c: the number of eigenvalues and eigenvectors to use for clustering. Set to 5 clusters.
- Proximity measure:

$$K(x,y) = \exp\left(-\frac{\text{dist}(x,y)^2}{\sigma^2}\right),$$

where `dist`(x,y) is the Euclidean distance.

- k: the number of nearest neighbors for the sparsification graph
- Graph Laplacian type: Use the standard Laplacian: $W - D$, where W is the proximity matrix

Jarvis-Patrick algorithm (SNN)

- Do not use a sparsification matrix. Instead, use the k-nearest neighbors (the proximity measure is not required) to determine the points used to compute the number of shared neighbors for each point.
- `smin`: number of shared neighbor threshold. Points below this threshold do not belong to a cluster .
- Distance metric: Euclidean.
- Create
- Run the Jarvis-Patrick algorithm as described in the code file. You should be able to create the graph using only the `numpy` module. Apply DBSCAN (create our own version with `numpy`) once you have computed the shared-nearest neighbor graph.

If you find a python implementation, make sure it does not use modules beyond those available to you i.e., `numpy`. For each method, you will play with two parameters specified in this assignment.

Some notes:

- You are given a dataset with 50,000 points that are mixed, which describe five clusters.
- Once you have found optimal parameters for your random sample of points, run each clustering algorithm five times and compute the SSE and ARI metric for each run. Compute the average and standard deviation of both metrics and return them in the answer dictionaries as specified in the template code.
- You only have access to the `numpy` module.
- For each algorithm, try *at least 10* combinations of parameter(s).

Describe the results found for each method. For each method, calculate the Adjusted Rand Index (ARI, `https://oecd.ai/en/catalogue/metrics/adjusted-rand-index-%28ari%29`. You'll return these in a dictionary.

Question 2: 50pts

In this problem, we explore the Expectation-Maximization (EM) algorithm, which generalizes Maximum Likelihood (ML). The dataset to model consists of 50,000 2-D points (x_i, y_i), $i = 0, 1, \ldots, 50,000$. We will consider a Gaussian Mixture model defined as the sum of two Gaussians:

$$P(x) = \rho_0 N(x_1, x_2; \mu_1, \Sigma_1) + \rho_1 N(x_1, x_2; \mu_2, \Sigma_2) \tag{1}$$

where

$$N(x_1, x_2; \mu, \Sigma) = \frac{1}{2\pi|\Sigma|^{1/2}} \exp\left(-(x-\mu)^T \Sigma^{-1} (x-\mu)\right) \tag{2}$$

and Σ is the 2D covariance matrix. Each sample point x_i has two components: x_{i_1} and x_{i_2}. The objective of this exercise is to estimate the amplitudes ρ_i, the two means $\mu_i = (\mu_{i1}, \mu_{i_2})$, and the two 2×2 symmetric covariance matrices Σ_i.

The function that will calculate the expectation maximization should be as follows (I provide the template):

```
    def em_algorithm(data: NDArray, max_iter: int=100)
        -> tuple(NDArray, NDArray, NDArray[NDArray], NDArray
        """
        Arguments:
        - data: numpy array of shape 50,000 x 2
        - max_iter: maximum number of iterations for the algorithm

        Return:
        - weights: the two coefficients \rho_1 and \rho_2 of the mixture model
        - means: the means of the two Gaussians (two scalars) as a list
        - covariances: the covariance matrices of the two Gaussians
          (each is a 2x2 symmetric matrix) return the full matrix
        - log_likelihoods: `max_iter` values of the log_likelihood,
          including the initial value

        Notes:
        - order the distribution parameters such that the x-component of
              the means are ordered from largest to smallest.
        - hint: the log-likelihood is monotonically increasing (or constant)
              if the algorithm is implemented correctly.
```

```
        - If this code is copied from some source, make sure to reference the
            source in this doc-string.
        """
        # CODE FILLED BY STUDENT
        return weights, means, covariances, log_likelihoods
```

You are provided with a data file with 50,000 rows with two floats in each column. The data file was created with `numpy.save()`. The data is read with the function `numpy.load()`. Create a function that estimates ρ, μ, and Σ for each distribution. Run the simulation ten times with ten subsets of 5000 points each (sample the points from the dataset with no repetition) and return this data in a format described in the source file. You are strongly encouraged to use AI to help you. I will be checking this during testing, and you will get a zero score for this question if you import *any* modules besides `numpy`, mainly because the testing software will fail. The returned parameters should be averaged over the 10 trials. Beware that parameters such as $\mu \in \mathbb{R}^2$, will appear in random order: sometimes (μ_{i1}, μ_{i2}) and sometime (μ_{i2}, μ_{i1}). This is handled inside the function **em_algorithm** function by ensuring that the order of the distribution parameters is such that the first component of the mean is in descending order. reorder them consistently before performing the average. Again, I suggest you use AI to help.

Return the 2×2 confusion matrix for this problem for each of the 10 slices, and return them in a dictionary of NDArrays (see source file). Return the ARI and SSE metrics for all 10 slices, along with its average and standard deviation. .

List of files provided

You can run each of these source codes on their own via (for example):

```
    python expectation_maximization.py
```

If there is a compilation error, it is extremely unlikely the automatic grader will work properly. If you have an error, and you are running out of time, I strongly suggest disabling the problematic part of the code to at least get partial credit.

Question1: Clustering

1. **expectation_maximization.py**
2. **question1_cluster_data.npy:**
3. **question1_cluster_labels.npy:**

Question2: Expectation Maximization

1. **spectral_clustering.py**
2. **denclue_clustering.py**
3. **jarvis_patrick_clustering.py**
4. **question2_cluster_data.npy:**
5. **question2_cluster_labels.npy:**

References

1. Class slides and Book chapter on Advanced Clustering (see Canvas).
2. Jarvis-Patrick: `https://www.geeksforgeeks.org/basic-understanding-of-jarvis-patrick-clusteri`
3. Spectral: `https://en.wikipedia.org/wiki/Spectral_clustering`