Homework 6

Deadline: December 19, 15:15

1. Principal Component Analysis (PCA) of Genomes (40 Points)

In this part, you will run PCA on a real dataset and interpret the output.
Download the p4dataset2023.txt file from the course website. The data is from the 1000 Genomes Project. Each of the 995 lines in the file represents an individual. The first three columns represent:

- the individual's unique identifier,
- their sex (`M` for male, `F` for female),
- the population they belong to (see the file p4dataset2023decoding.txt for decoding these population tags).

The subsequent 10,101 columns of each line are a subsample of nucleobases from the individual's genome. We will analyze the output of PCA on this dataset. When referring to "PCA" in this section, we mean:

- The data should be centered (i.e., the sample mean subtracted out) but not normalized, so it is acceptable if some dimensions have different variances.
- The output should be the normalized principal components (i.e., unit-length vectors).

Feel free to use a library implementation of PCA, such as scikit-learn's implementation. Note that with Python's `scikit-learn`, you can specify how many principal components you want, which can save computation time.
First, convert the data from the text file of nucleobases to a real-valued matrix (PCA requires a real-valued matrix). Specifically, convert the genetic data into a binary matrix X such that:

$$X_{i,j} = \begin{cases} 0 & \text{if the } i\text{th individual has column } j\text{'s mode nucleobase for their } j\text{th nucleobase,} \\ 1 & \text{otherwise.} \end{cases}$$

Note that all mutations appear as a 1, even if they are different mutations. For example, if the mode for column j is "G", then if individual i has an "A", "T", or "C", $X_{i,j}$ would be 1. Ignore the first three columns of the data file when creating the binary matrix X. We will examine genotypes to extract phenotype information.

Q1 Say we ran PCA on the binary matrix X above. What would be the dimension of the returned vectors? [2 Points]

Q2 Examine the first two principal components of X. These components contain significant information about our dataset. Create a scatter plot with each of the 995 rows of X projected onto the first two principal components. In other words:

- The horizontal axis should be v_1.
- The vertical axis should be v_2.
- Each individual should be projected onto the subspace spanned by v_1 and v_2.

Your plot must use a different color for each population and include a legend. [8 Points]

Q3 In two sentences, list one or two basic facts about the plot created in part (b). Can you interpret the first two principal components? What aspects of the data do the first two principal components capture? *Hint: Think about history and geography.* [10 Points]

Q4 Examine the third principal component of X. Create another scatter plot with each individual projected onto the subspace spanned by the first and third principal components. After plotting, play with different labeling schemes (with labels derived from the meta-data) to explain the clusters you observe. Your plot must include a legend. [6 Points]

Q5 What information does the third principal component capture? Write a one-sentence answer. [8 Points]

Q6 Inspect the third principal component. Plot the nucleobase index versus the absolute value of the corresponding value of the third principal component in that index. (The x-axis of your plot should go from 1 to 10,101—you're literally just plotting the 10,101 values in the third principal component.) What do you notice? What's a possible explanation? *Hint: Think about chromosomes.* [6 Points]

2. Clustering with k-Means and EM (40 Points)

This exercise requires you to implement the k-means and Expectation-Maximization (EM) algorithms for clustering a synthetic dataset. You are provided with a partially implemented Python file E2.ipynb, which includes:

- **Data generation**: A synthetic 2D dataset with Gaussian clusters.
- **Plotting utilities**: A function to visualize the dataset and clustering results at different iterations.

- **Class definitions**: Definitions for:
  - `KMeansCustom` (for k-means clustering)
  - `EMGMM` (for Expectation-Maximization)

Your task is to fill in the missing functions in the provided class definitions to complete the implementation of the k-means and EM algorithms. You will then use these implementations to analyze clustering performance and compare the results.

Exercises

Q1. Data Visualization

Run the provided E2.ipynb file to generate and visualize the synthetic 2D dataset. The dataset consists of 2000 points from two Gaussian clusters with the following parameters:

- **Means:** $[-1, -1]$ and $[1, 1]$
- **Covariances:** $\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$
- **Proportions:** 0.5 for each cluster

Verify that the ground-truth cluster labels and means are plotted correctly.

Q2. Implement k-Means Clustering (10 Points)

Complete the implementation of the following functions in the `KMeansCustom` class:

1. `assign_clusters`: Assign each data point to the nearest centroid.
2. `update_centroids`: Update centroids as the mean of all points assigned to each cluster.

Test your implementation by running k-means for 20 iterations and visualizing the clustering progress at iterations 0, 1, 2, 5, 10, and 20. For each visualization:

- Use different colors to represent cluster assignments.
- Mark centroids with a distinct symbol (e.g., a black "X").

Q3. Implement EM Algorithm for GMM (10 Points)

Complete the implementation of the following functions in the `EMGMM` class:

1. `e_step`: Compute the responsibilities (soft assignments) for each cluster.

2. `m_step`: Update the means, covariances, and weights based on the computed responsibilities.

3. `multivariate_gaussian`: Compute the probability density of a multivariate Gaussian distribution for a given data point.

Run the EM algorithm for 20 iterations and visualize the clustering progress at iterations 0, 1, 2, 5, 10, and 20. For each visualization:

- Use different colors to represent soft assignments.
- Overlay ellipses representing the Gaussian covariances for each cluster.

Q4. Compare and Analyze (10 Points)

1. Compute the Adjusted Rand Index (ARI) for the final cluster assignments from both k-means and EM. Which algorithm performs better? Why?

2. Discuss the strengths and limitations of each algorithm:
   - How does k-means handle overlapping clusters compared to EM?
   - How do the assumptions of Gaussian distributions in EM affect its performance?

3. Discuss the visual differences in cluster assignments at intermediate iterations for both algorithms.

Q5. The Effect of Overestimating Clusters (5 Points)

1. What happens when the number of clusters k is overestimated (e.g., $k > 2$)? simulate and plot the results.

2. (optional) How can we estimate the optimal number of clusters? Discuss methods like the Elbow Method, Silhouette Scores, or Bayesian Information Criterion (BIC).

Q6. Non-Circular Clusters (5 Points)

1. Generate a synthetic dataset with two non-circular clusters using Scikit-learn's `make_moons` function.

2. Apply k-means and EM to this dataset. Visualize the clustering results for $k = 2$ and $k = 4$.

3. Discuss the performance of both algorithms on non-circular clusters. Which algorithm performs better, and why?

4. (optional) Can you suggest a method to fix this?

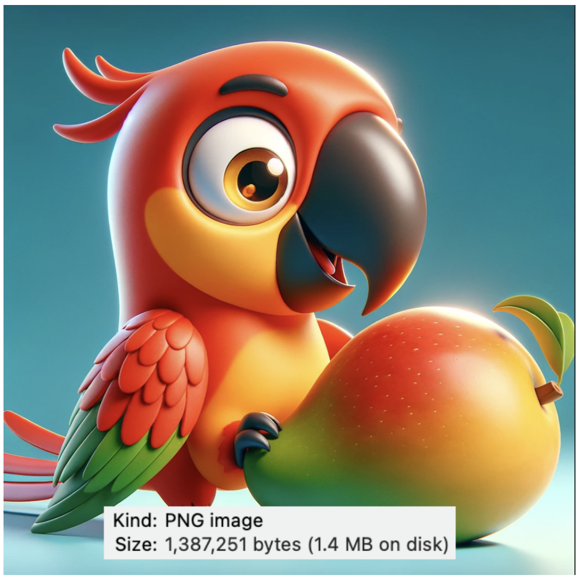

Figure 1: Macaw Image

3. Image Segmentation with k-Means Clustering (20 Points)

In this task, you will use k-means clustering to perform image segmentation on a color image. The goal is to cluster pixel intensities in the RGB space into k groups and create a simplified version of the image with reduced colors.

Q1. Load and Visualize the Image

1. Load the provided color image file Macaw.webp shown in Figure 1.
2. Visualize the original image using Matplotlib.

You can use the following Python code snippet to load and display the image:

```python
import matplotlib.pyplot as plt
from skimage import io

# Load the color image
image_path = "original_image.jpg"
image = io.imread(image_path)

# Display the image
plt.imshow(image)
plt.title("Original Image")
plt.axis("off")
plt.show()
```

Q2. Apply k-Means Clustering for Image Segmentation (15 Points)

1. Flatten the image into a 2D array where each row represents a pixel's RGB values.
2. Use k-means clustering to group pixels into $k = 2$, $k = 4$, and $k = 8$ clusters in the RGB space.
3. Replace each pixel's RGB values with the centroid of its cluster to create segmented images.
4. Visualize:
   - The original color image.
   - The segmented images for $k = 2$, $k = 4$, and $k = 8$.

You may use the k-means implementation you developed in the previous exercise or any other k-means implementation such as Scikit-learn's implementation to perform clustering.

Q3. Analyze the Results (5 Points)

Answer the following questions:

1. How does increasing k (number of clusters) affect the segmented image?
2. What are the trade-offs between using a small and a large k?