Worksheet 7: Soft-clustering

Soft Clustering

We generate 10 data points that come from a normal distribution with mean 5 and variance 1.

```
import random
import numpy as np
from sklearn.cluster import KMeans

mean = 5
stdev = 1

s1 = np.random.normal(mean, stdev, 10).tolist()
print(s1)
```

a. Generate 10 more data points, this time coming from a normal distribution with mean 8 and variance 1.

```
s2 = np.random.normal(8, 1, 10).tolist()
print(s2)
```

b. Flip a fair coin 10 times. If the coin lands on H, then pick the last data point of `s1` and remove it from `s1`, if T then pick the last data point from `s2` and remove it from `s2`. Add these 10 points to a list called `data`

```
data = []
for i in range(10):
    coin_output = random.choice([0, 1])  # Flip coin (0 = H, 1 = T)
    if coin_output == 0:
        p1 = s1.pop()  # Take last element of s1
        data.append(p1)
    else:
        p2 = s2.pop()  # Take last element of s2
        data.append(p2)
```

```
print(data)
```

c. This `data` is a Gaussian Mixture Distribution with 2 mixture components. Over the next few questions we will walk through the GMM algorithm to see if we can uncover the parameters we used to generate this data. First, please list all these parameters of the GMM that created `data` and the values we know they have.

Mean and Variance of each Gaussian component

- μ1 = 5, σ12 = 1 (for s1)
- μ2 = 8, σ22=1 (for s2)

Mixing Coefficients (π)

- Since we flipped a **fair coin**, each data point had a **50% probability** of coming from s1 or s2.
- So, the **mixing coefficients** (weights) are:
  - π1=0.5 (probability of choosing s1)
  - π2=0.5 (probability of choosing s2)

d. Let's assume there are two mixture components (note: we could plot the data and make the observation that there are two clusters). The EM algorithm asks us to start with a random `mean_j`, `variance_j`, `P(S_j)` for each component j. One method we could use to find sensible values for these is to apply K means with k=2 here.

   a. the centroids would be the estimates of the `mean_j`
   b. the intra-cluster variance could be the estimate of `variance_j`
   c. the proportion of points in each cluster could be the estimate of `P(S_j)`

   Go through this process and list the parameter estimates it gives. Are they close or far from the true values?

```
from sklearn.cluster import KMeans
import numpy as np

kmeans = KMeans(2, init='k-means++', random_state=42).fit(np.array(dat
```

```
s1 = [x[0] for x in filter(lambda x: x[1] == 0, zip(data, kmeans.labels_))]
s2 = [x[0] for x in filter(lambda x: x[1] == 1, zip(data, kmeans.labels_))]

prob_s = [len(s1) / (len(s1) + len(s2)), len(s2) / (len(s1) + len(s2))]
mean = [sum(s1) / len(s1), sum(s2) / len(s2)]
var = [
    sum((x - mean[0])**2 for x in s1) / len(s1),
    sum((x - mean[1])**2 for x in s2) / len(s2)
]

print(f"P(S_1) = {prob_s[0]}, P(S_2) = {prob_s[1]}")
print(f"mean_1 = {mean[0]}, mean_2 = {mean[1]}")
print(f"var_1 = {var[0]}, var_2 = {var[1]}")
```

e. For each data point, compute $P(S_j \mid X_i)$. Comment on which cluster you think each point belongs to based on the estimated probabilities. How does that compare to the truth?

```
from scipy.stats import norm

prob_s0_x = []  # P(S_0 | X_i)
prob_s1_x = []  # P(S_1 | X_i)

for p in data:
    pdf_i = [norm.pdf(p, mean[0], np.sqrt(var[0])), norm.pdf(p, mean[1], np.

    # Compute total probability P(X_i)
    prob_x = prob_s[0] * pdf_i[0] + prob_s[1] * pdf_i[1]

    # Compute posterior probabilities using Bayes' rule
    prob_s0_x.append(prob_s[0] * pdf_i[0] / prob_x)
    prob_s1_x.append(prob_s[1] * pdf_i[1] / prob_x)

# Print results
for i, p in enumerate(data):
    print(f"Point {p}:")
    print(f"  P(S_1 | X) = {prob_s0_x[i]:.4f}")
    print(f"  P(S_2 | X) = {prob_s1_x[i]:.4f}\n")
```

f. Having computed `P(S_j | X_i)`, update the estimates of `mean_j`, `var_j`, and `P(S_j)`. How different are these values from the original ones you got from K means? briefly comment.

```
prob_s = [sum(prob_s0_x) / len(prob_s0_x), sum(prob_s1_x) / len(prob_s1_x

mean = [
    sum(p * w for p, w in zip(data, prob_s0_x)) / sum(prob_s0_x),
    sum(p * w for p, w in zip(data, prob_s1_x)) / sum(prob_s1_x)
]

var = [
    sum(w * (p - mean[0])**2 for p, w in zip(data, prob_s0_x)) / sum(prob_s
    sum(w * (p - mean[1])**2 for p, w in zip(data, prob_s1_x)) / sum(prob_s1
]

print(f"P(S_1) = {prob_s[0]}, P(S_2) = {prob_s[1]}")
print(f"mean_1 = {mean[0]}, mean_2 = {mean[1]}")
print(f"var_1 = {var[0]}, var_2 = {var[1]}")
```

g. Update `P(S_j | X_i)`. Comment on any differences or lack thereof you observe.

```
prob_s0_x = []
prob_s1_x = []

for p in data:
    pdf_i = [norm.pdf(p, mean[0], np.sqrt(var[0])), norm.pdf(p, mean[1], np.
    prob_x = prob_s[0] * pdf_i[0] + prob_s[1] * pdf_i[1]

    prob_s0_x.append(prob_s[0] * pdf_i[0] / prob_x)
    prob_s1_x.append(prob_s[1] * pdf_i[1] / prob_x)

for i, p in enumerate(data):
    print(f"Point {p}:")
    print(f"  Updated P(S_1 | X) = {prob_s0_x[i]:.4f}")
    print(f"  Updated P(S_2 | X) = {prob_s1_x[i]:.4f}\n")
```

h. Use `P(S_j | X_i)` to create a hard assignment - label each point as belonging to a specific cluster (0 or 1)

```
assignments = [0 if prob_s0_x[i] > prob_s1_x[i] else 1 for i in range(len(data

for i, p in enumerate(data):
    print(f"Point {p}: Assigned to cluster {assignments[i]}")
```