Homework 5

SHREYA CHETAN PAWASKAR
pawaskas@uci.edu
12041645

Question 1:

a)

- For Data A, where the two point sets are well-separated and have a circular shape, the EM algorithm should produce the given clustering correctly.
- However, for the other three data sets, the EM algorithm would generate substantially different clusterings from the given one
    - For Data B, instead of the given clustering, the EM algorithm would group the points in the lower-left region into one circular cluster and the points in the upper-right region into another circular cluster.
    - For Data C, rather than the given clustering, the EM algorithm would cluster the left half of both point sets into one group and the right half of both point sets into another group.
    - For Data D, instead of the given clustering, the EM algorithm would put the lower half of both point sets into one cluster and the upper half of both point sets into another cluster.
- EM works well for circular and oval-shaped groups (Data A and C).
- EM struggles with square-like shapes (Data B and D)

---

b)

- For Data A, where the two clusters have a circular shape, EM should produce the given clustering correctly.
- For Data C, EM should also produce the given clustering, as it can learn that the clusters have larger variance in the x1 dimension and smaller variance in the x2 dimension.
- However, EM would produce substantially different clusterings for Data B and Data D:
    - For Data B, which has an open-square shape, the diagonal-covariance Gaussians used by EM cannot capture the positive correlations in the data. Therefore, EM would merge some points from the different clusters together, resulting in a different clustering.
    - For Data D, even though the data has a non-convex open-square shape, EM would prefer to split it into multiple clusters instead of modeling it as a single cluster. This is because EM finds it challenging to model non-convex shapes with a single Gaussian distribution.

---

c)

- EM should produce the given clusterings for Data A, B, and C, as these datasets have elliptical shapes.
- For Data D, which has an open-square arrangement, the Gaussian model used by EM is a poor fit.
- Instead of the given clustering for Data D, EM would model the data differently:
    - The lower half of the points would be modeled with a positively correlated Gaussian distribution.
    - The upper half of the points would be modeled with a negatively correlated Gaussian distribution.

Question 2:

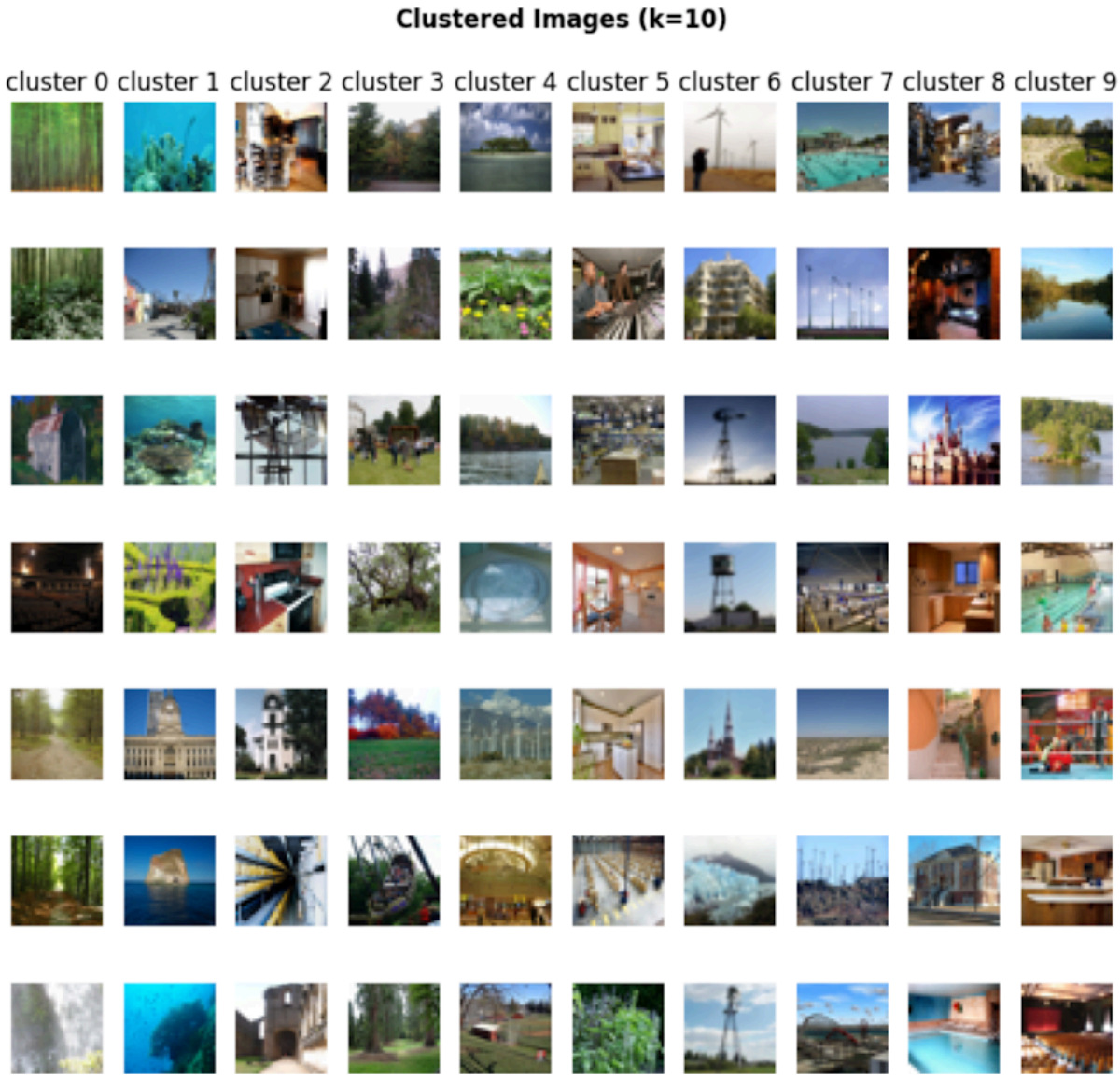
Code:

```
!pip install pomegranate
```

```
import numpy as np
import random
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import pickle as pk
from pomegranate import *
from em_utils import *
from general_utils import *

# Load the data
data = np.load("subset_of_sun397_tiny_dataset_with_pca.npy",
allow_pickle=True).item()
images = data["images"]
labels = data["labels"]
pca_features = data["pca_features"].astype(np.float32)

# Load the trained PCA
pca = pk.load(open("pca.pkl", 'rb'))

# The parameters to try
num_clusters_to_try = [10, 20]
figures = []

for k in num_clusters_to_try:
   print("")
   print("{:0d} Clusters".format(k))

   # Create the initial cluster assignments
   initial_cluster_assignments = create_initial_clusters(pca_features, k)

   # Ensure components are created with float32
   components = create_initial_cluster_distributions(pca_features.astype(np.float32),
k, initial_cluster_assignments)

   # Do the EM algorithm
   model = CustomGeneralMixtureModel(components, tol=1e-8, verbose=False)
```

```
    model.fit(pca_features)

    fig0 = plot_clustered_images(model, pca_features, images)
    fig1 = plot_sampled_images(model, pca)

    figures.append((fig0, "clustered_images_{:02d}_clusters".format(k)))
    figures.append((fig1, "sampled_images_{:02d}_clusters".format(k)))
plt.show()
```

A) k=10



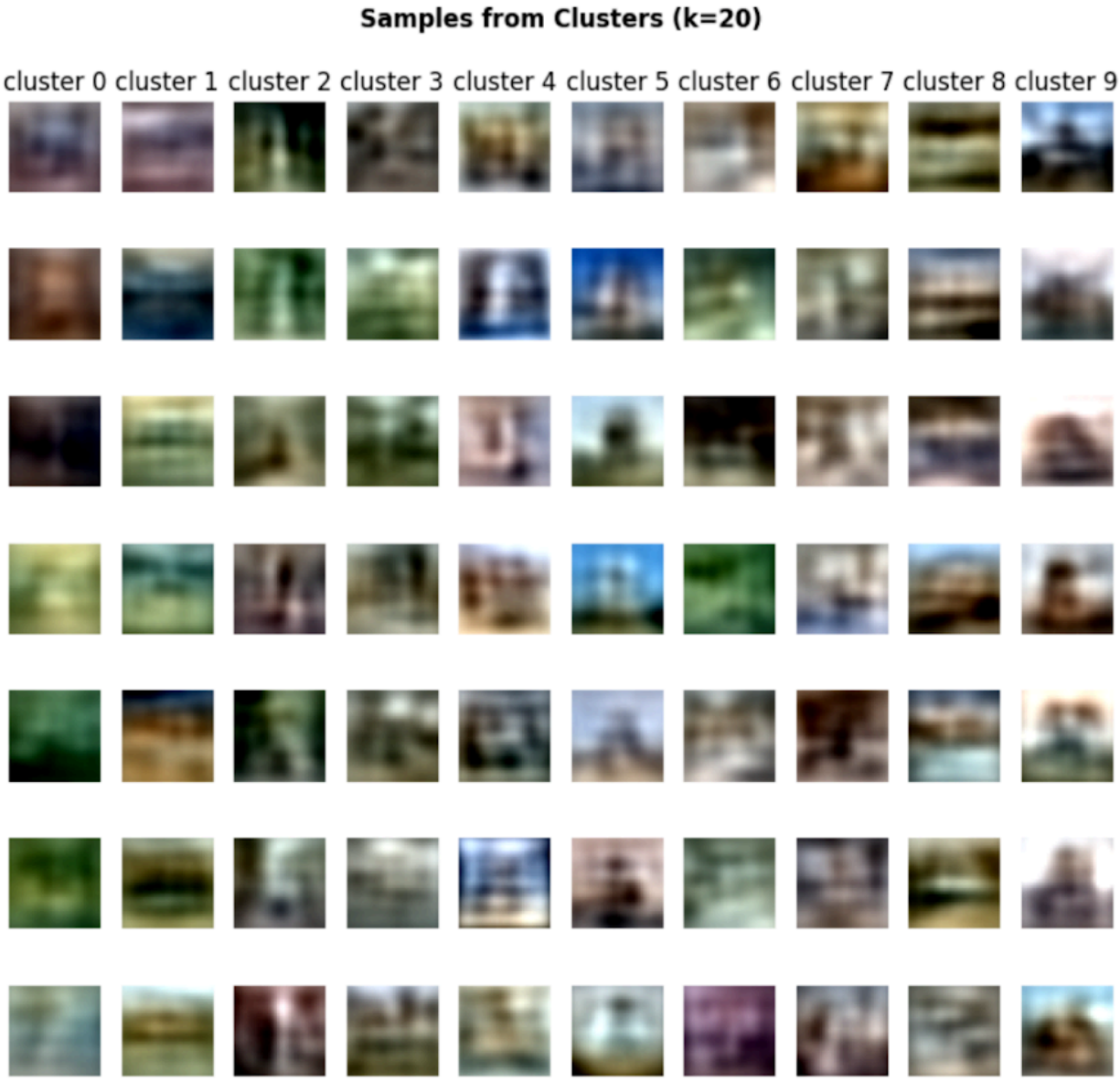
When the number of clusters K is set to 10, the images appear to be primarily clustered based on their dominant colors. One cluster contains images with a significant amount of green, likely representing grass, vegetation, and other outdoor greenery. Another cluster consists of images with blue hues, which could be capturing scenes of the sky or bodies of water. Additionally, there seems to be a cluster that groups together images featuring prominent central buildings or towers as the

main subject. The clustering algorithm has effectively identified and separated the images based on these broad color and subject characteristics.

B) k=20

See Figure 2 for examples of images with high probability of being assigned to various clusters when K = 20. Again we see that images are clustered based on color and if there is a large central object in the image. We also see that the clusters capture more scene information. We can see that cluster 1 in Figure 2 mainly contains buildings and indoor spaces, while cluster 3 is of large open outdoor spaces. Clustering with K = 20 generally seems to yield more coherent clusters, whose images are more closely related.

C)



- See Figure for samples from the learned Gaussian clusters, which capture the color of the images well as well as some coarse shape features. In cluster 4 we can see that the images have a bright streak down the center (almost like a waterfall).
- In cluster 3 we can see that the images look like they have 2 parts: sky and ground. Cluster 1 samples seem to often show (blurry) foreground objects: large color blobs are present, though it is not clear what type of object is visible. The multivariate Gaussian clusters don't model fine detail well, nor do they capture specifics of a scene such as indoor vs outdoor or building vs no building.
- The multivariate Gaussian clusters mainly capture very general information like color, sky, and if the scene is open or more cluttered.

Question 3:

Code:

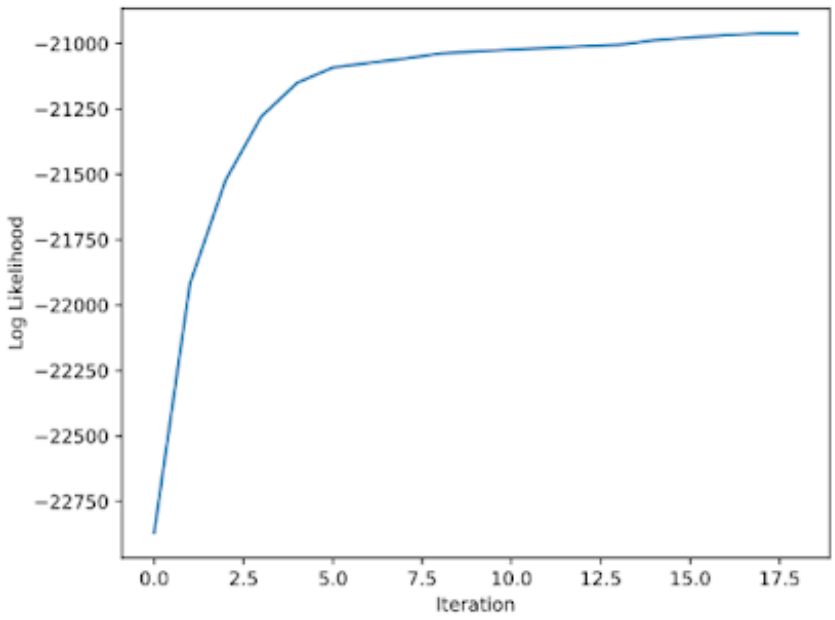
```
import numpy as np
import matplotlib.pyplot as plt
import random
from pomegranate import *

def create_random_components(x, number_of_clusters):
   # Randomize the order of the images and the labels since they are not shuffled
   rand_indices = list(range(x.shape[0]))
   random.shuffle(rand_indices)
   shuffled_x = x[rand_indices]

   components = []
   split_size = shuffled_x.shape[0] // number_of_clusters

   for k in range(number_of_clusters):
       # Extract some data for the split
       s = k * split_size
       e = s + split_size
       data = shuffled_x[s:e]

       # Compute the MLE for the Bernoulli from data.
       # Add random pseudo-count to the value and clamp it to be in range
       p = np.mean(data, axis=0)
       p += np.random.randn(p.shape[0]) * 0.1
       p[p < 0] = 0.0
       p += 0.001
       p[p > 1] = 1.0

       # Create the distribution
       dist = FixedBernoulli(probs=p)
       components.append(dist)

   return components

def save_top_words(model, x, labels, num_words=5):
   # Compute the log probs for each of the samples for each of the clusters
   log_probs = model.predict_log_proba(x)
```

```
    number_of_clusters = log_probs.shape[-1]

    print("\n\nTop Words:")
    for k in range(number_of_clusters):
        print(f"\nNumber of Clusters = {k}")

        # We want the max so do the negative log prob
        sorted_indices = np.argsort(-log_probs[:, k])
        selected_indices = sorted_indices[:num_words]
        selected_labels = labels[selected_indices]

        for sl in selected_labels:
            print(sl.item())

def save_top_ranked_features(model, labels, number_of_dims, num_features=5):
    number_of_clusters = 8

    # Extract the parameters for each of internal distributions
    all_cluster_params = np.zeros((number_of_clusters, number_of_dims))

    for c in range(number_of_clusters):
        cluster_dists = model.distributions[c]
        all_cluster_params[c, :] = cluster_dists.get_p_values_numpy()

    print("Top Features:")

    for k in range(number_of_clusters):
        print(f"Number of Clusters = {k}")

        # Grab top features for cluster k
        cluster_params = all_cluster_params[k]
        sorted_indices = np.argsort(-cluster_params)
        selected_indices = sorted_indices[:num_features]
        selected_feat_labels = labels[selected_indices]

        for x in selected_feat_labels:
            print(x.item())

data = np.load("wordcat.npy", allow_pickle=True).item()
x_train = data["Xtrain"]
x_labels = data["Xlabels"]
```

```
feat_labels = data["featLabels"]

# Need to subtract 1 since x_train is [1, 2] but we need [0, 1]
x_train -= 1
number_of_dims = x_train.shape[-1]

number_of_clusters = 8
components = create_random_components(x_train, number_of_clusters)

bmm = CustomGeneralMixtureModel(components, tol=1e-8, verbose=True)
_, history = bmm.fit(x_train)

log_probs = bmm.predict_log_proba(x_train)

plt.figure(figsize=(8, 6))
plt.plot(history[1:], linewidth=2, color='blue')
plt.xlabel("Iteration" , fontsize=14)
plt.ylabel("Log Likelihood", fontsize=14)
plt.show()

number_of_clusters = 8
models = []
final_log_probs = []

for i in range(10):
   components = create_random_components(x_train, number_of_clusters)
   bmm = CustomGeneralMixtureModel(components, tol=1e-8, verbose=True)
   _, history = bmm.fit(x_train)

   final_log_prob = history[-1]
   models.append(bmm)
   final_log_probs.append(final_log_prob)

print("Best Model")
best_model_idx = final_log_probs.index(max(final_log_probs))
best_model = models[best_model_idx]
save_top_words(best_model, x_train, x_labels)
save_top_ranked_features(best_model, feat_labels, number_of_dims)

print("Worst Model")
worst_model_idx = final_log_probs.index(min(final_log_probs))
```

```
worst_model = models[worst_model_idx]
save_top_words(worst_model, x_train, x_labels)
save_top_ranked_features(worst_model, feat_labels, number_of_dims)
```

a)



1) When the pseudocount parameter is set to 0, the expectation-maximization (EM) algorithm computes a maximum likelihood estimate, which assumes a uniform prior distribution. However, if the pseudocount is changed to any other value, the EM algorithm instead computes a maximum a posteriori (MAP) estimate, which incorporates a non-uniform Dirichlet prior distribution.
2) During the EM iterations, the log-likelihood value should continually increase because each step optimizes a lower bound on the marginal log-likelihood.
3) The plot in Figure 4 demonstrates this expected behavior, where the log-likelihood monotonically increases as the algorithm progresses.
4) When implementing the EM algorithm, verifying that the log-likelihood is consistently increasing across iterations is an extremely useful debugging tool.
5) If the log-likelihood fails to increase or decrease, it may indicate an issue with the implementation or the convergence of the algorithm.

```
Best Model

Top Words:

Number of Cluster = 0  -jar, hose, church, cathedral, skillet

Number of Cluster = 1  - goat, chipmunk, buzzard, emu, owl

Number of Cluster = 2  - cabin, table, pencil, coin, skyscraper
```

```
Number of Cluster = 3  - cheetah, lobster, doll, panther, lion

Number of Cluster = 4  - motorcycle, van, rake, shield, shotgun

Number of Cluster = 5  - trousers, helmet, nightgown, cap_(hat), shack

Number of Cluster = 6  - machete, bazooka, scissors, hammer, pipe_(plumbing)

Number of Cluster = 7  - banana, cantaloupe, yam, blueberry, raisin

Worst Model

Top Words:
Number of Cluster = 0  -whale platypus guppy certificate magazine

Number of Cluster = 1  - Ship shawl cloak cape bench

Number of Cluster = 2 - Zucchini vine faucet asparagus sandals

Number of Cluster = 3  - Trousers broom carrot guitar pants

Number of Cluster = 4  - Oven mouse van blender sailboat

Number of Cluster = 5  - Goat plug_(electric) sparrow robin birch

Number of Cluster = 6  - Emu bluejay rattlesnake building pie

Number of Cluster = 7  - Cheetah deer leopard cat tomato
```

b)

- The model with the highest likelihood (best run) produces clusters where the words within each mixture component are coherent and semantically related.
- For example, component 6 clearly represents tools and component 1 represents animals.
- The worst run of EM has some reasonable clusters, but also seems to split related words across clusters.
- The worst run places words like "oven" and "blender" in the same cluster as seemingly unrelated words like "sailboat" and "mouse."

---

c)

```
Best Model:
Top Features:

Number of Cluster = 0 - made_of_plastic made_of_metal is_round found_in_kitchens
made_of_wood

Number of Cluster = 1  - an_animal beh_-_flies is_small has_wings has_a_beak

Number of Cluster = 2 - made_of_wood is_small lives_in_water made_of_metal beh_-_swims
```

```
Number of Cluster = 3 - an_animal is_large made_of_metal is_electrical a_mammal

Number of Cluster = 4 - made_of_metal is_large is_loud a_musical_instrument made_of_wood

Number of Cluster = 5 - different_colours clothing is_long made_of_metal worn_by_women

Number of Cluster = 6  - made_of_metal a_weapon is_dangerous has_a_handle is_sharp

Number of Cluster = 7  - is_edible a_vegetable is_green tastes_good a_fruit

Worst Model:
Top Features:
Number of Cluster = 0-  is_large beh_-_swims lives_in_water made_of_metal made_of_wood

Number of Cluster = 1 - Clothing different_colours made_of_metal is_long a_weapon

Number of Cluster = 2  - is_edible is_small is_white a_vegetable eaten_by_cooking

Number of Cluster = 3 - is_long made_of_wood a_musical_instrument inbeh_-_produces_music
different_colours

Number of Cluster = 4 - made_of_metal has_a_handle is_small made_of_plastic
used_for_transportation

Number of Cluster = 5 - is_small an_animal beh_-_flies has_wings beh_-_eats

Number of Cluster = 6 - a_bird an_animal has_feathers has_a_beak has_wings

Number of Cluster = 7 - an_animal is_edible is_small a_fruit has_4_legs
```

- The best run appears slightly more interpretable than the worst run.
- For example, the best model learns to cluster living things together (Cluster 4). However, it still struggles to cluster in Cluster 3, where something made_of_metal is paired together with an animal.
- The worst model really struggles to cluster things together, e.g., an animal is paired with a fruit in Cluster 7.