M11-L2 Problem 1

In this problem you will implement the elbow method using three different sklearn clustering algorithms: (`KMeans` , `SpectralClustering` , `GaussianMixture`). You will use the algorithms to find the number of natural clusters for two different datasets, one "blob" shaped dataset, and one concetric circle dataset.

In [241…

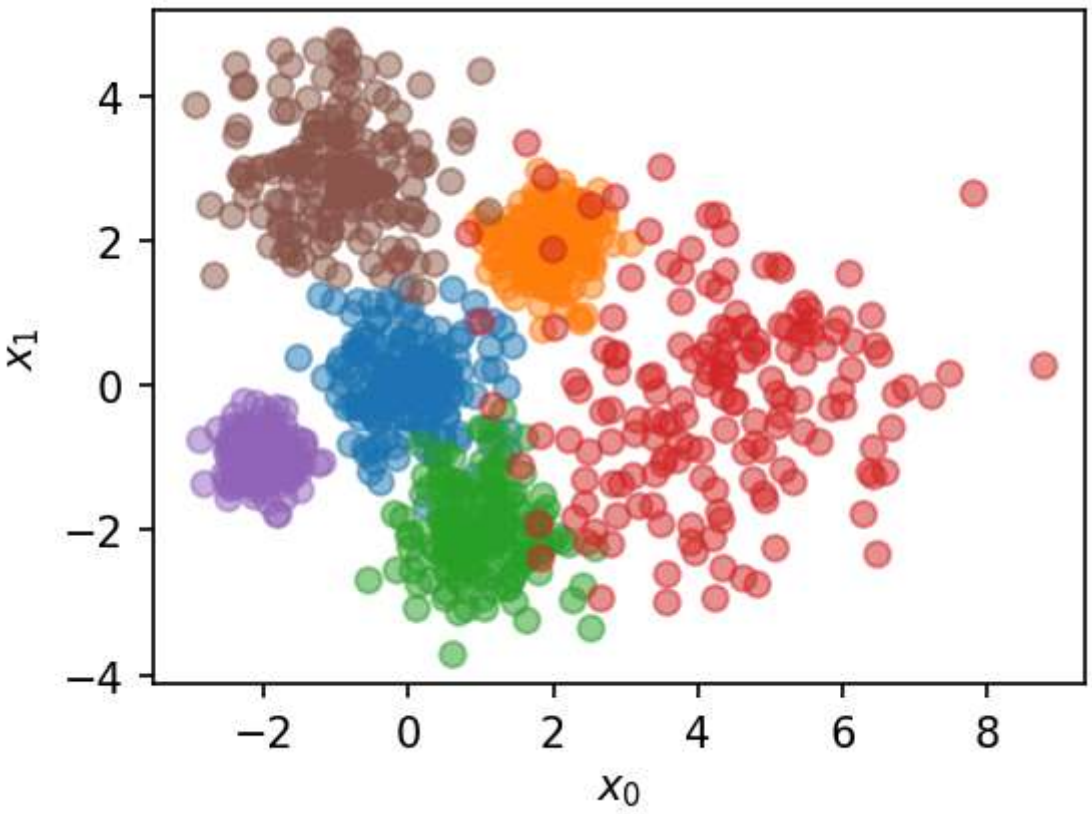
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200

from sklearn.datasets import make_blobs, make_circles
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.mixture import GaussianMixture

def plot_loss(loss, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    ax.plot(np.arange(2, len(loss)+2), loss, 'k-o')
    ax.set_xlabel('Number of Clusters')
    ax.set_ylabel('Loss')
    if title:
        ax.set_title(title)
    return ax

def plot_pred(x, labels, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    n_clust = len(np.unique(labels))
    for i in range(n_clust):
        ax.scatter(x[labels == i,0], x[labels == i,1], alpha = 0.5)
    ax.set_title(title)
    return ax

def compute_loss(x, labels):
    # Initialize loss
    loss = 0
    # Number of clusters
    n_clust = len(np.unique(labels))
    # Loop through the clusters
    for i in range(n_clust):
        # Compute the center of a given label
        center = np.mean(x[labels == i, :], axis = 0)
        # Compute the sum of squared distances between each point and its correspon
        loss += np.sum(np.linalg.norm(x[labels == i, :] - center, axis = 1)**2)
    return loss
```

Blob dataset

Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where `y` contains the label of each corresponding point in `x`.

In [242…
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 1000, n_features = 2, centers = [[0,0],[2,2],[1,-2],[
```

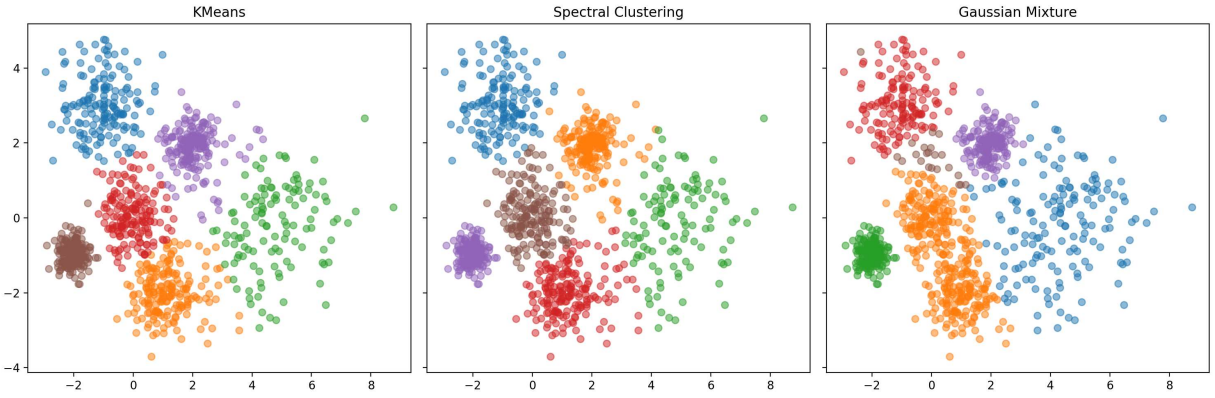
In [243…
```
## YOUR CODE GOES HERE
def plotter(x, labels = None, ax = None, title = None):
    if ax is None:
        _, ax = plt.subplots(dpi = 150, figsize = (4,4))
        flag = True
    else:
        flag = False
    for i in range(len(np.unique(labels))):
        ax.scatter(x[labels == i, 0], x[labels == i, 1], alpha = 0.5)
    ax.set_xlabel('$x_0$')
    ax.set_ylabel('$x_1$')
    ax.set_aspect('equal')
    if title is not None:
        ax.set_title(title)
    if flag:
        plt.show()
    else:
        return ax
plotter(x,y)
```



Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the "blob" data with 6 clusters, and modify the parameters until

you get satisfactory results. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

In [ ]:
```
## YOUR CODE GOES HERE
fig, ax = plt.subplots(1, 3, figsize=(15, 5), sharey=True)



model1 = KMeans(n_clusters=6,n_init=1,algorithm="elkan")
model1.fit(x)
labels1 = model1.labels_
centers = model1.cluster_centers_
plot_pred(x,labels1,ax[0],title = 'KMeans')


model2 = SpectralClustering(n_clusters=6,n_init=1)
model2.fit(x)
labels2 = model2.labels_
plot_pred(x,labels2,ax[1],title = 'Spectral Clustering')

model3 = GaussianMixture(n_components=6,init_params="random_from_data")
model3.fit(x)
pred = model3.predict(x)
plot_pred(x,pred,ax[2],title = 'Gaussian Mixture')

plt.tight_layout()
plt.show()
```

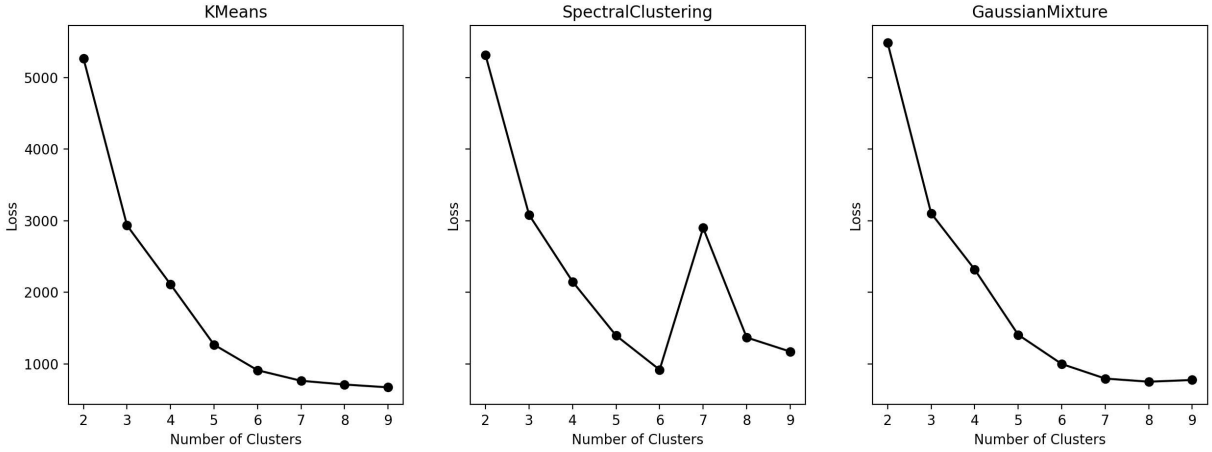


Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_loss(x, labels, ax, title)` function.

In [245…
```
## YOUR CODE GOES HERE
n_clust = [2,3,4,5,6,7,8,9]
Losses1 = np.array([])
Losses2 = np.array([])
Losses3 = np.array([])
```

```
for i in n_clust:
    model1 = KMeans(n_clusters=i,n_init=1)
    model1.fit(x)
    labels1 = model1.labels_
    centers = model1.cluster_centers_
    loss1 = compute_loss(x,labels1)
    Losses1 = np.append(Losses1,loss1)


    model2 = SpectralClustering(n_clusters=i,n_init=1)
    model2.fit(x)
    labels2 = model2.labels_
    loss2 = compute_loss(x,labels2)
    Losses2 = np.append(Losses2,loss2)


    model3 = GaussianMixture(n_components=i)
    model3.fit(x)
    labels3 = model3.predict(x)
    loss3 = compute_loss(x,labels3)
    Losses3 = np.append(Losses3,loss3)

fig, ax = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
plot_loss(Losses1,ax=ax[0], title=f'KMeans')
plot_loss(Losses2,ax=ax[1], title=f'SpectralClustering')
plot_loss(Losses3,ax=ax[2], title=f'GaussianMixture')
```

Out[245… `<Axes: title={'center': 'GaussianMixture'}, xlabel='Number of Clusters', ylabel='Loss'>`



Concentric circles dataset

Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where `y` contains the label of each corresponding point in `x`.

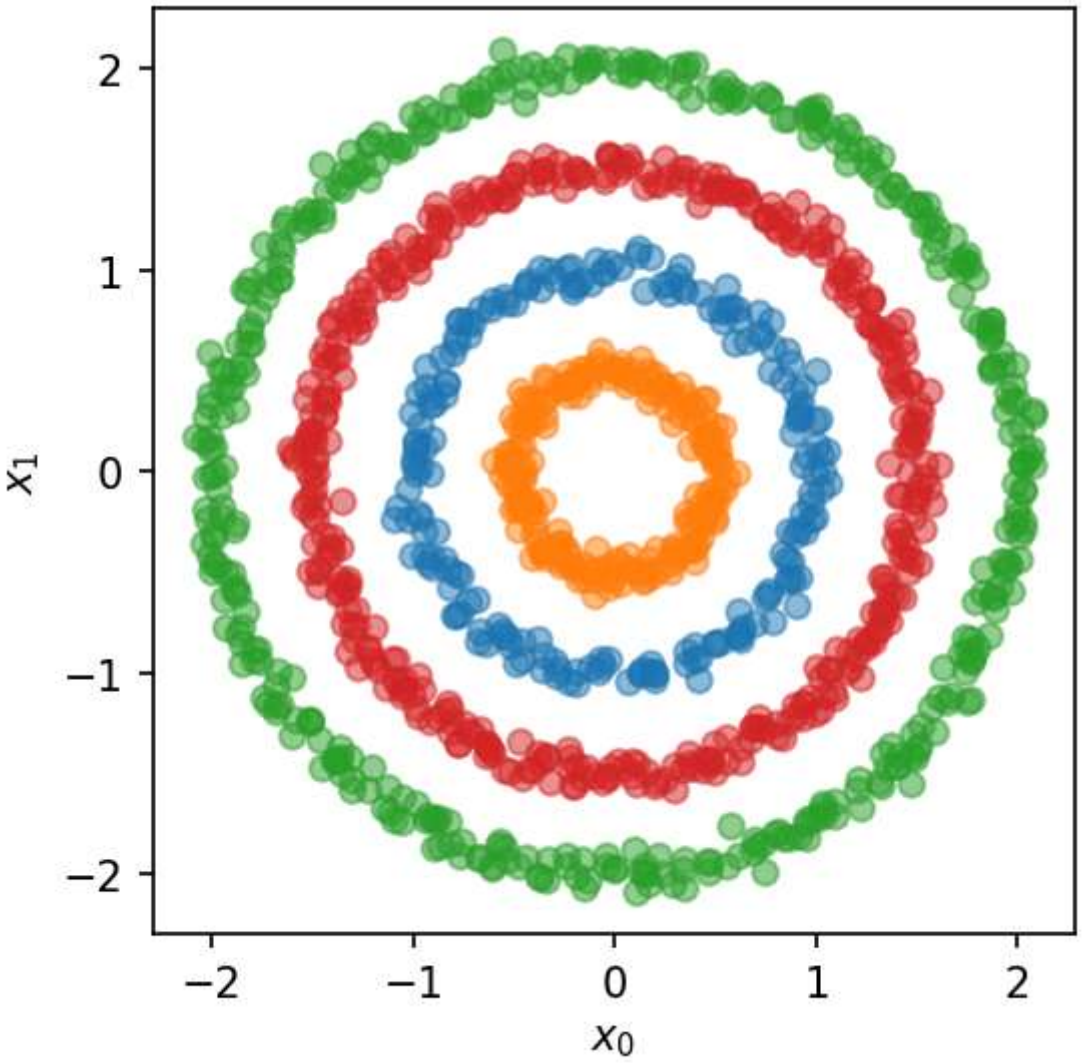
In [246…

```
## DO NOT MODIFY
x1, y1 = make_circles(n_samples = 400, noise = 0.05, factor = 0.5, random_state = 0
x2, y2 = make_circles(n_samples = 800, noise = 0.025, factor = 0.75, random_state =
```

```
x = np.vstack([x1, x2*2])
y = np.hstack([y1, y2+2])
```

In [247…

```
## YOUR CODE GOES HERE
plotter(x,y)
```



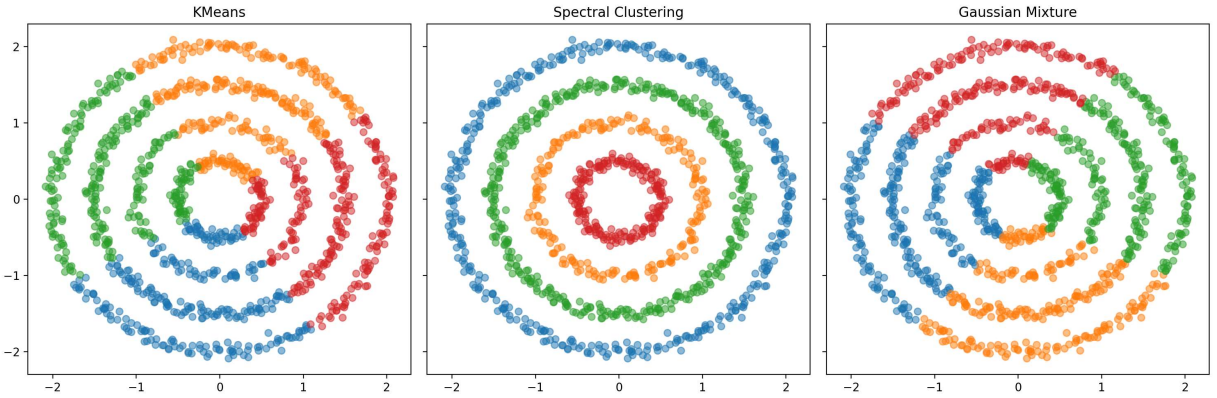
Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the concentric circle data with 4 clusters, and attempt to modify the parameters until you get satisfactory results. Note: you should get good clustering results with Spectral Clustering, but the KMeans and GMM models will struggle to cluster this dataset well. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

In [ ]:

```
## YOUR CODE GOES HERE
fig, ax = plt.subplots(1, 3, figsize=(15, 5), sharey=True)

model1 = KMeans(n_clusters=4,n_init=1)
model1.fit(x)
labels1 = model1.labels_
```

```
centers = model1.cluster_centers_
plot_pred(x,labels1,ax[0],title = 'KMeans')


model2 = SpectralClustering(n_clusters=4,n_init=1,affinity='nearest_neighbors')
model2.fit(x)
labels2 = model2.labels_
plot_pred(x,labels2,ax[1],title = 'Spectral Clustering')

model3 = GaussianMixture(n_components=4)
model3.fit(x)
pred = model3.predict(x)
plot_pred(x,pred,ax[2],title = 'Gaussian Mixture')

plt.tight_layout()
plt.show()
```

```
C:\Users\zsqu4\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\manifold\_spectral_embedding.py:329: UserWarning: Graph is not fully connected, spectral embedding may not work as expected.
  warnings.warn(
```



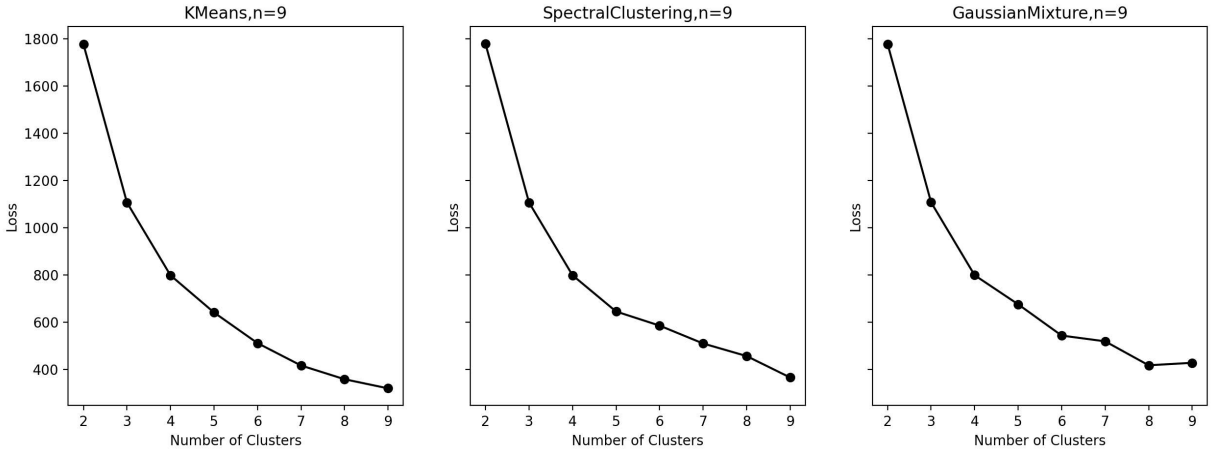
Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_loss(x, labels, ax, title)` function.

In [249…

```
## YOUR CODE GOES HERE
n_clust = [2,3,4,5,6,7,8,9]
Losses1 = np.array([])
Losses2 = np.array([])
Losses3 = np.array([])


for i in n_clust:
    model1 = KMeans(n_clusters=i,n_init=1)
    model1.fit(x)
    labels1 = model1.labels_
    centers = model1.cluster_centers_
```

```
    loss1 = compute_loss(x,labels1)
    Losses1 = np.append(Losses1,loss1)


    model2 = SpectralClustering(n_clusters=i,n_init=1)
    model2.fit(x)
    labels2 = model2.labels_
    loss2 = compute_loss(x,labels2)
    Losses2 = np.append(Losses2,loss2)


    model3 = GaussianMixture(n_components=i)
    model3.fit(x)
    labels3 = model3.predict(x)
    loss3 = compute_loss(x,labels3)
    Losses3 = np.append(Losses3,loss3)

fig, ax = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
plot_loss(Losses1,ax=ax[0], title=f'KMeans,n={i}')
plot_loss(Losses2,ax=ax[1], title=f'SpectralClustering,n={i}')
plot_loss(Losses3,ax=ax[2], title=f'GaussianMixture,n={i}')
```

Out[249… `<Axes: title={'center': 'GaussianMixture,n=9'}, xlabel='Number of Clusters', ylabel='Loss'>`



Discussion

1. Discuss the performance of the clustering algorithms on the "blob" dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods? Does the elbow method work better for some algorithms versus others?

Your response goes here\

KMeans and GaussianMixture perform pretty well as the loss descends pretty quick and converges to a low level. The SpectralClustering method has less ideal results compare to the other two.
Using the elbow method, the number of natural cluster is likely 6 or 7, as loss

decrease significantly slower after that. The elbow method works the better for KMeans and GaussianMixture, but there's some fluxuation in Spectural clustering.

2. Discuss the performance of the clustering algorithms on the concentric circles dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods?

Your response goes here
Using the elbow method, the number of natural cluster is likely 4 or 5, as loss decrease significantly slower after that. The elbow method does not really work here for any of the methods as there's not a super clear point that the improvement drastically decreases.

3. Does the sum of squared distances work well as a loss function for each of the three clustering algorithms we implemented? Does the sum of squared distance fail on certain types of clusters?

Your response goes here

The algorithm works fairly well for all three models in blob dataset. However, for concentric circle dataset, the algorithm failed to capture the pattern for the dataset for KMean and GaussianMixture.