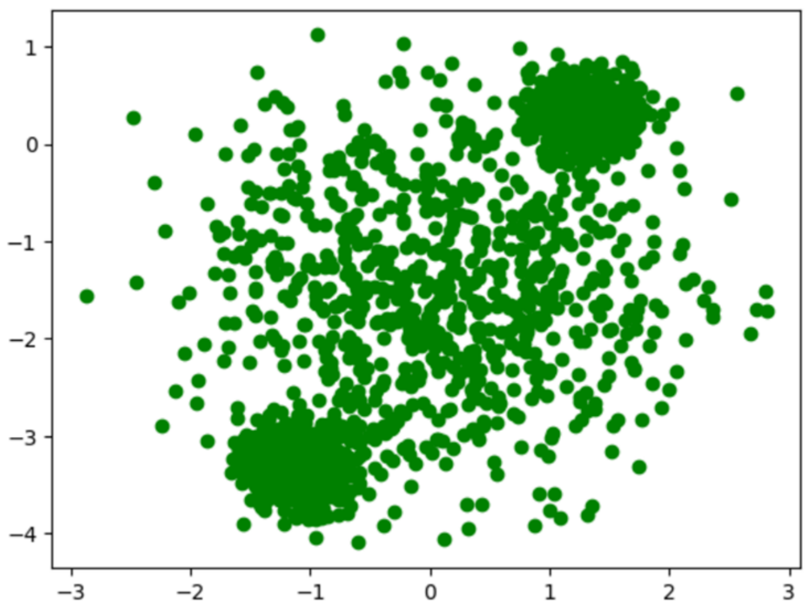
ECE 421 Programming Assignment 4 Question
Member One: Mark Qi
Student ID: 1006764645
Member Two: Richard Zhao
Student ID: 1006750614

Part 1: K-means Clustering

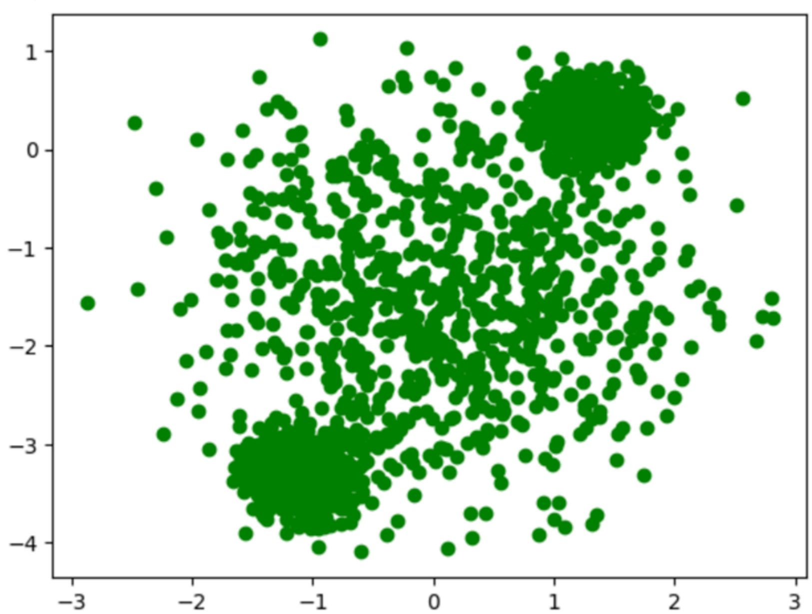
K = 1

Run 1

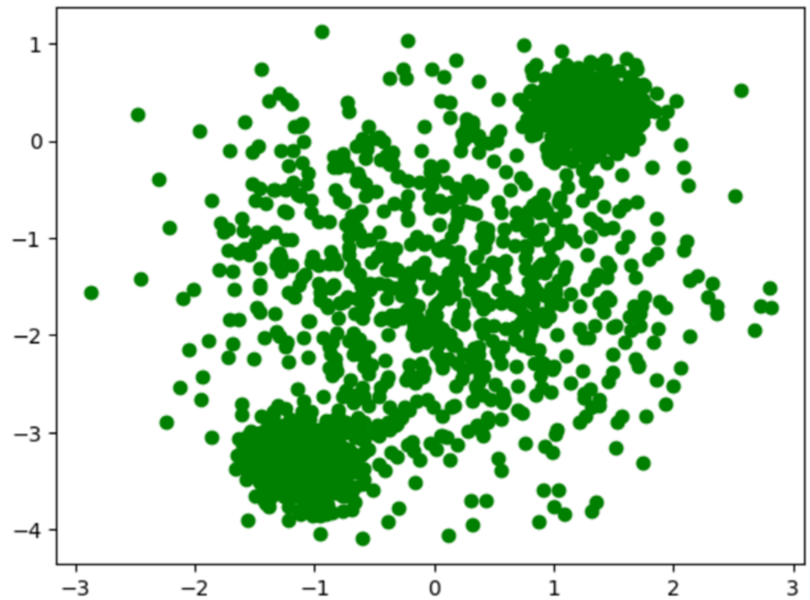
Pytorch score = 3.886242617244746



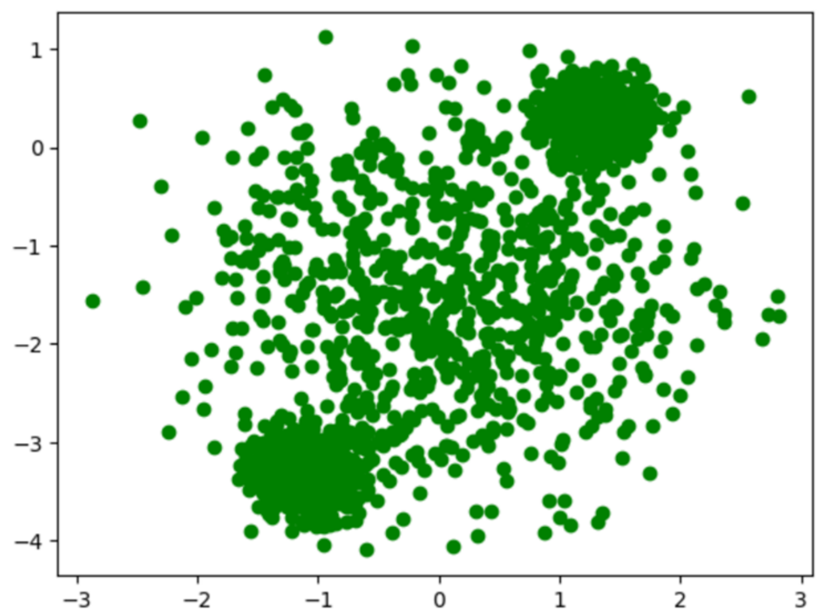
Scikit, score = 3.8862500723240965



Run 2
Pytorch score = 3.886207489392007



Scikit, score = 3.8862500723240965

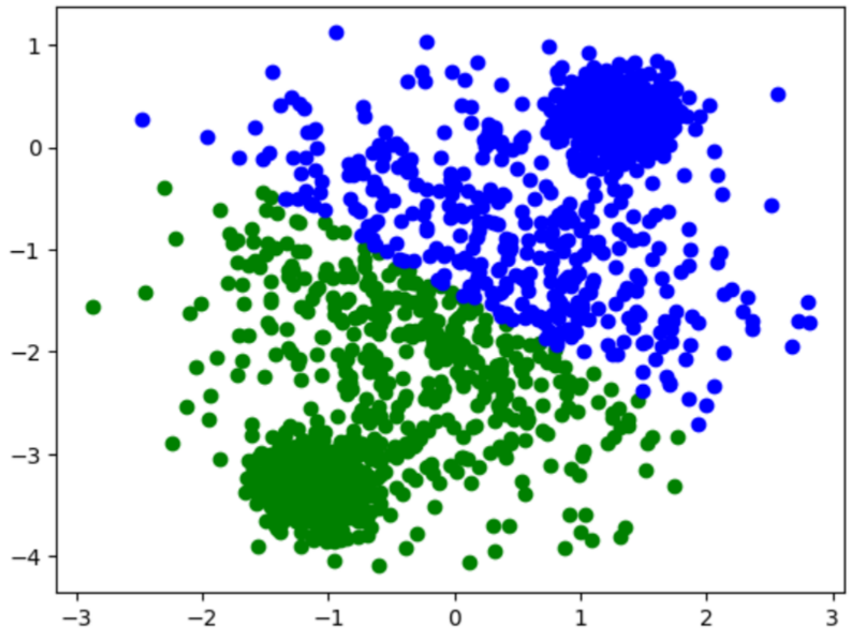


Comment: when k = 1, Pytorch scores are different for different runs, which means the clustering centers are different due to the random initialization. The Scikit scores are the same for different runs, and the score is generally higher than that of Pytorch.

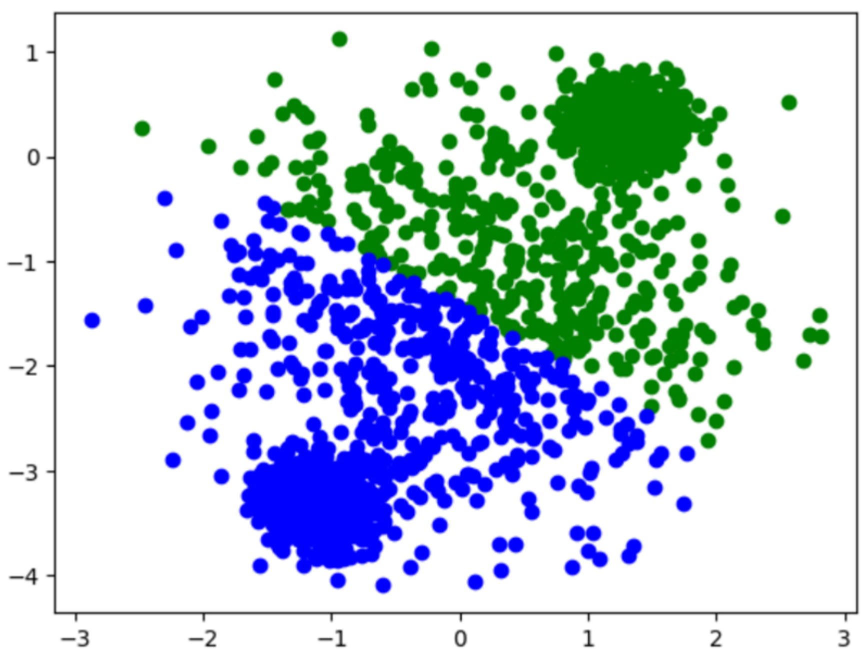
K = 2

Run 1

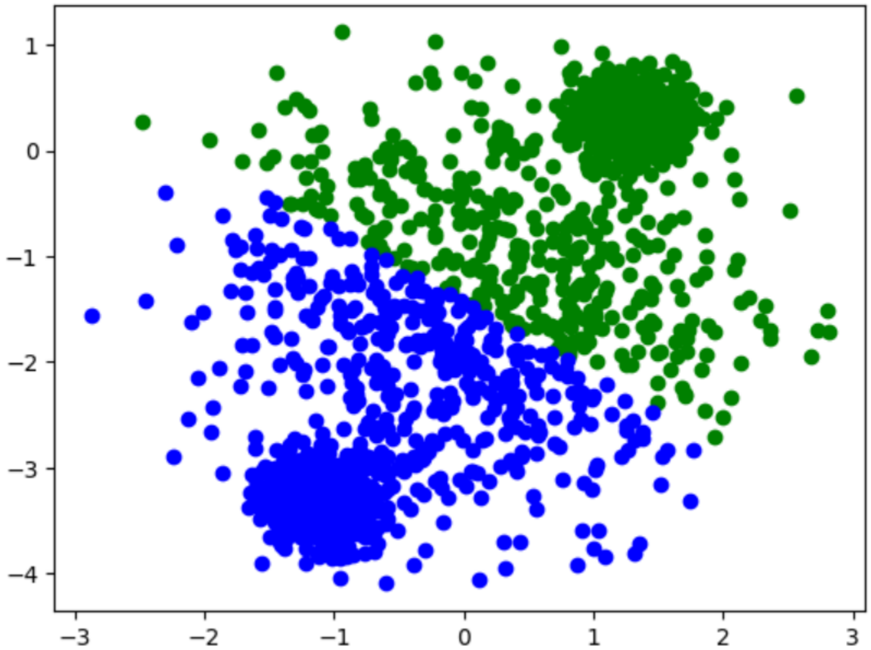
Pytorch, score = 0.874908158654531



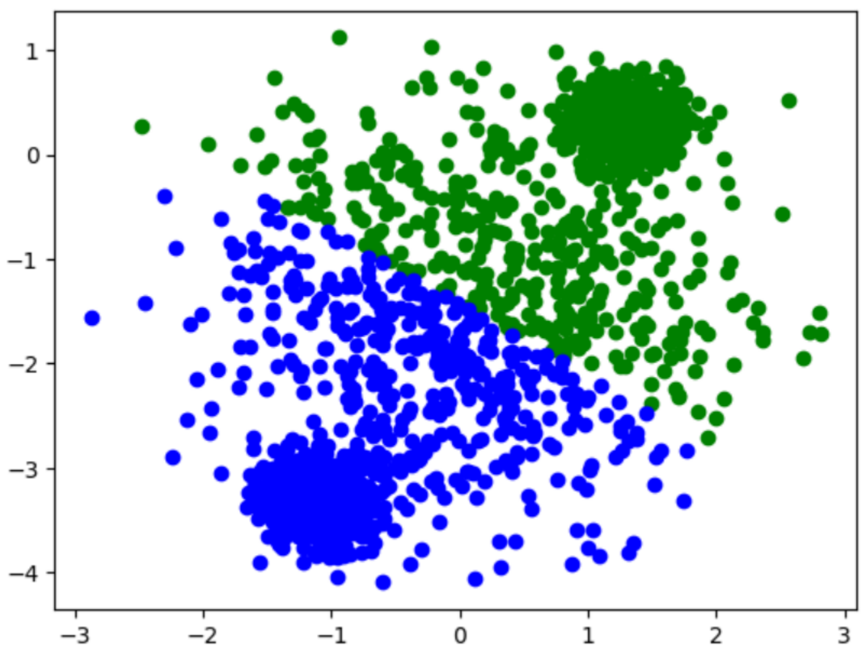
Scikit, score = 0.8749609567839518



Run 2
Pytorch, score = 0.8749099213712747



Scikit, score = 0.8749609567839518

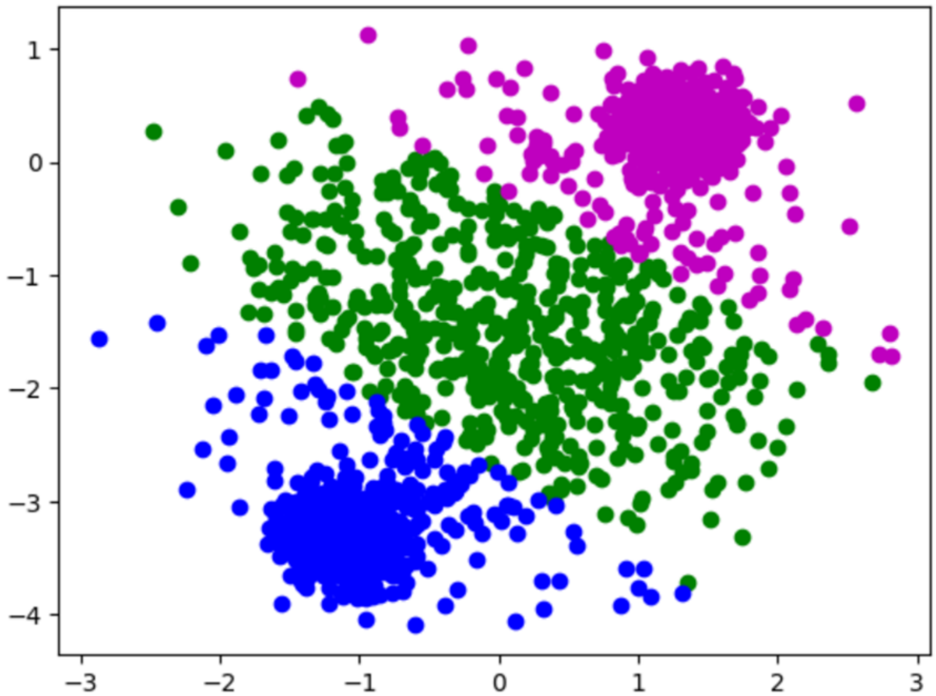


Comment: when k = 2, Pytorch scores are different for different runs, which means the clustering centers are different due to the random initialization. The Scikit scores are the same for different runs, and the score is generally higher than that of Pytorch. The clustering graphs look almost the same for Pytorch and Scikit for different runs.

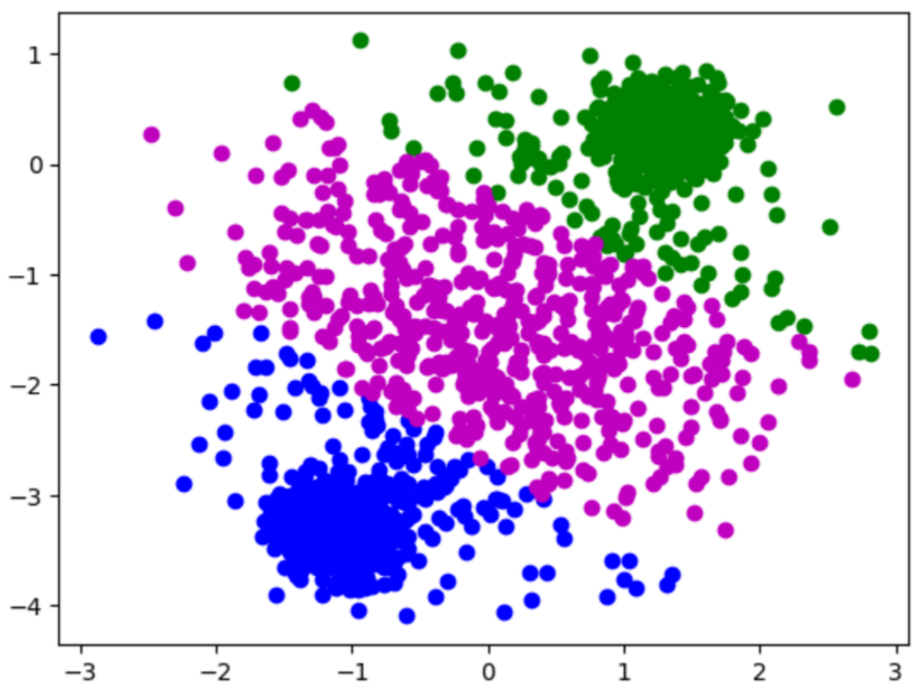
K = 3

Run 1

Pytorch, score = 0.48852354694432215

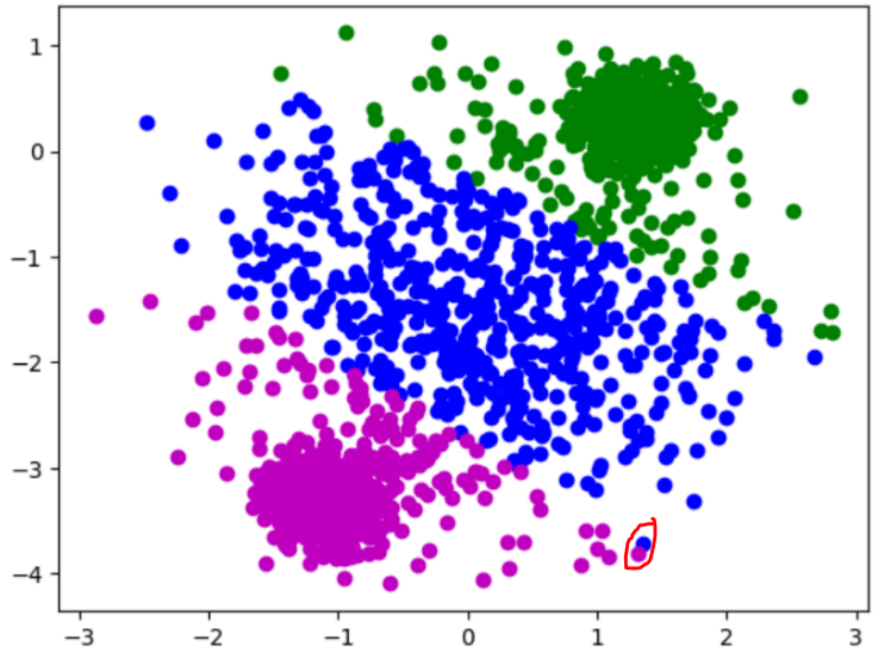


Scikit, score = 0.488244936700977

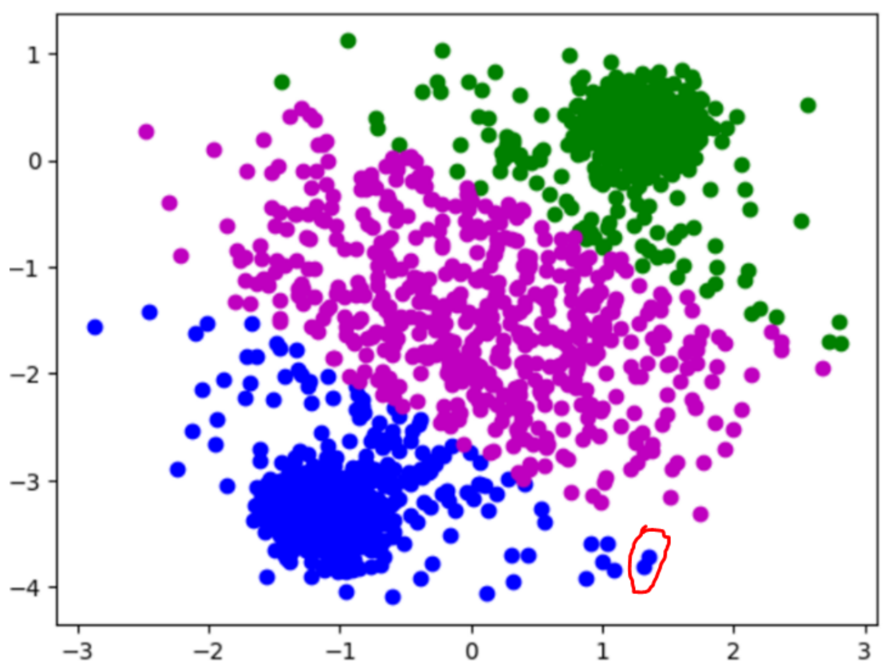


Run 2

Pytorch, score = 0.4885468051094047



Scikit, score = 0.488244936700977

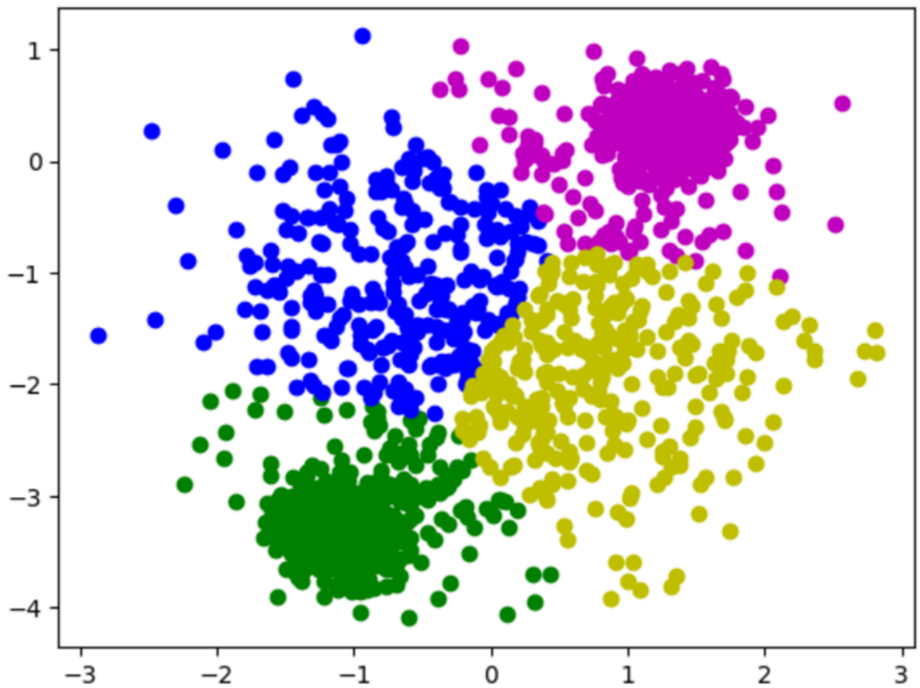


Comment: when k = 3, the conclusion is similar to the above comments (Pytorch produces different results across different runs), but the Pytorch scores are generally higher than the Scikit scores. Also, there are small differences in the clustering graph between Scikt and Pytorch this time, see the red circles above.

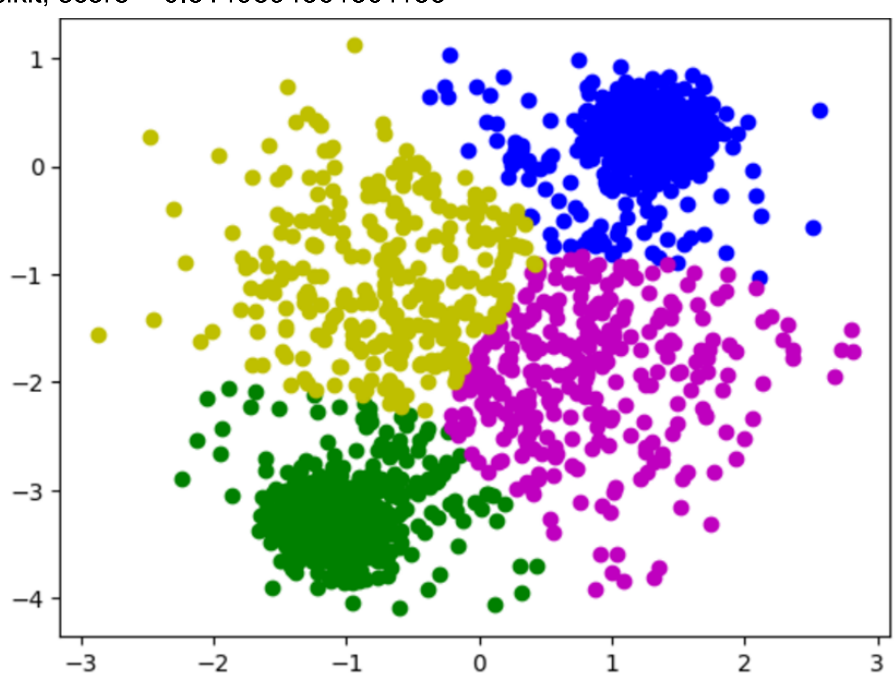
K = 4

Run 1

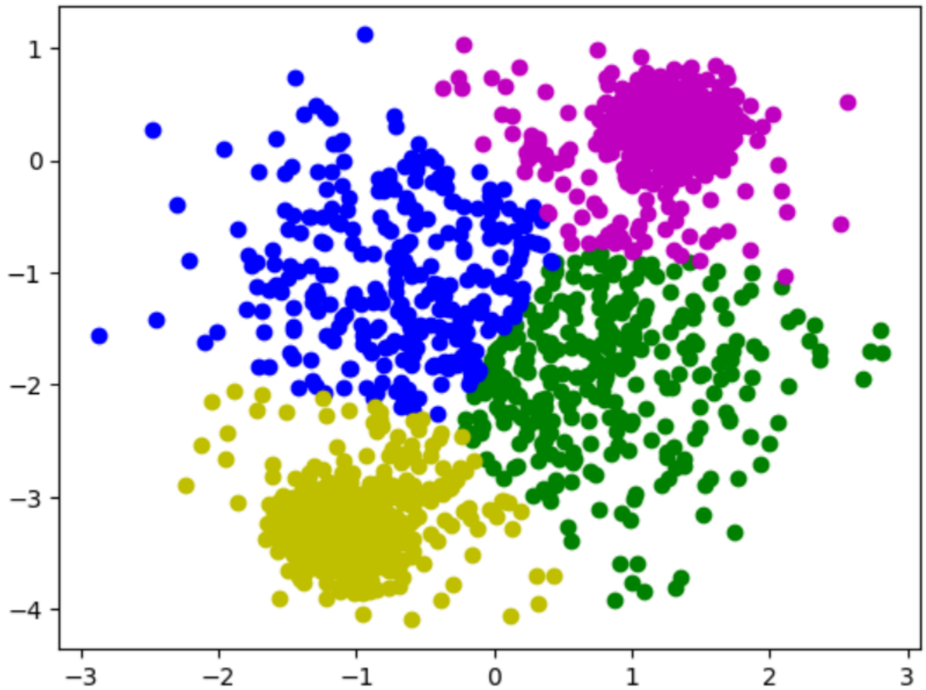
Pytorch, score = 0.31514094901418377



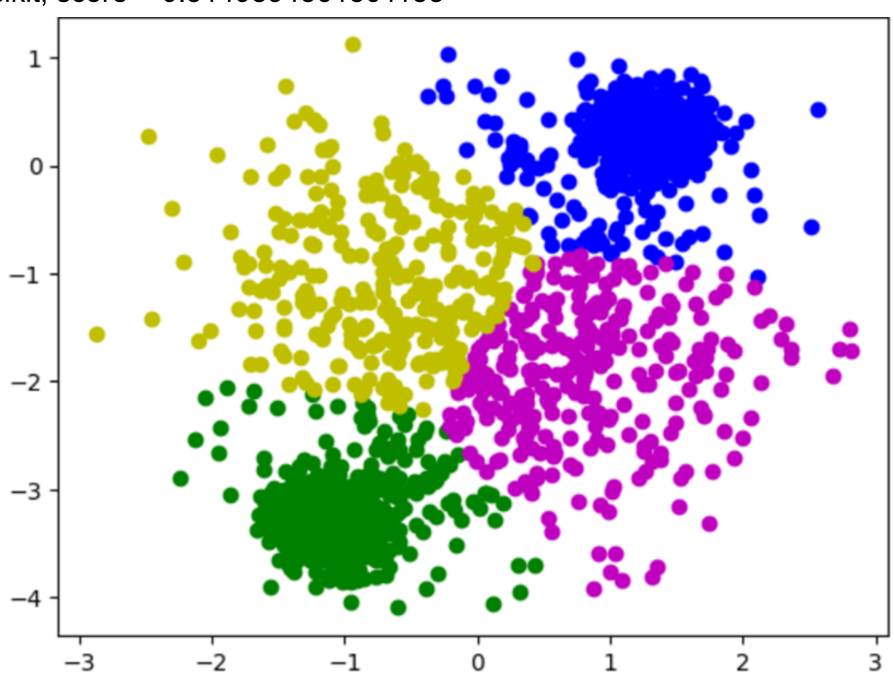
Scikit, score = 0.3149894361501133



Run 2
Pytorch, score = 0.31514806764571596



Scikit, score = 0.3149894361501133

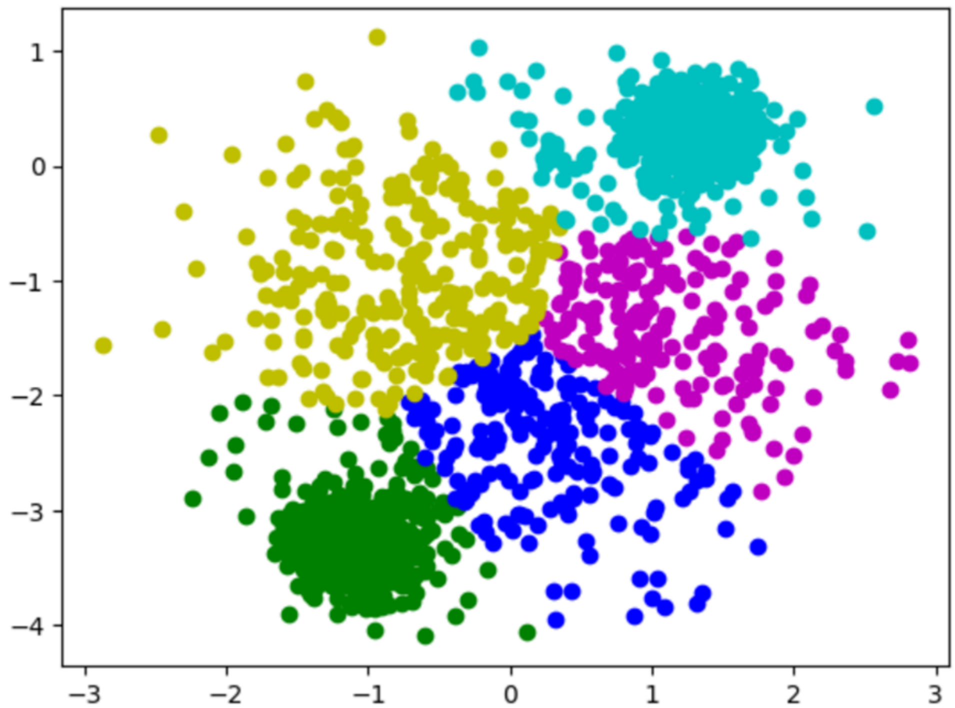


Comment: when k = 4, the conclusion is similar to that case in K = 4 (different Pytoch scores, etc), but the Pytorch scores are a lot higher than the Scikit scores now.

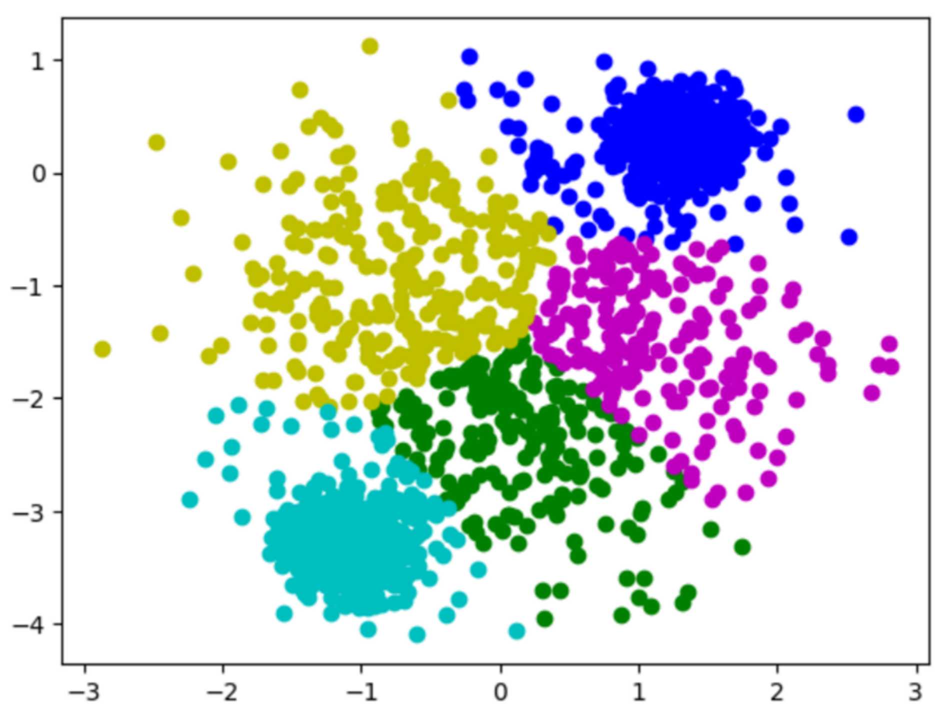
K = 5

Run 1

Pytorch, score = 0.26437155722958716

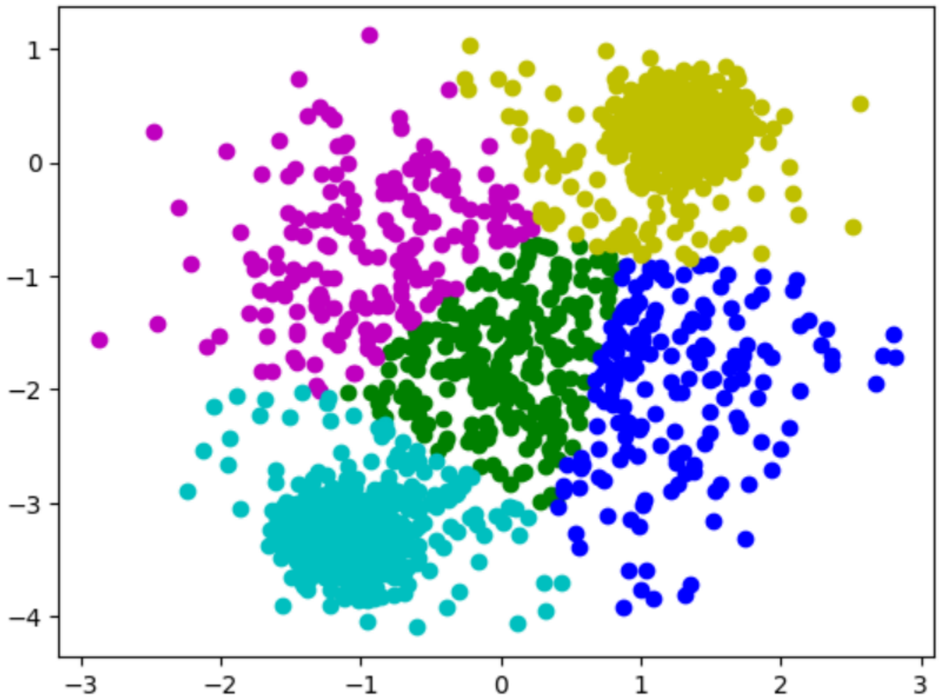


Scikit, score = 0.26462592838204324

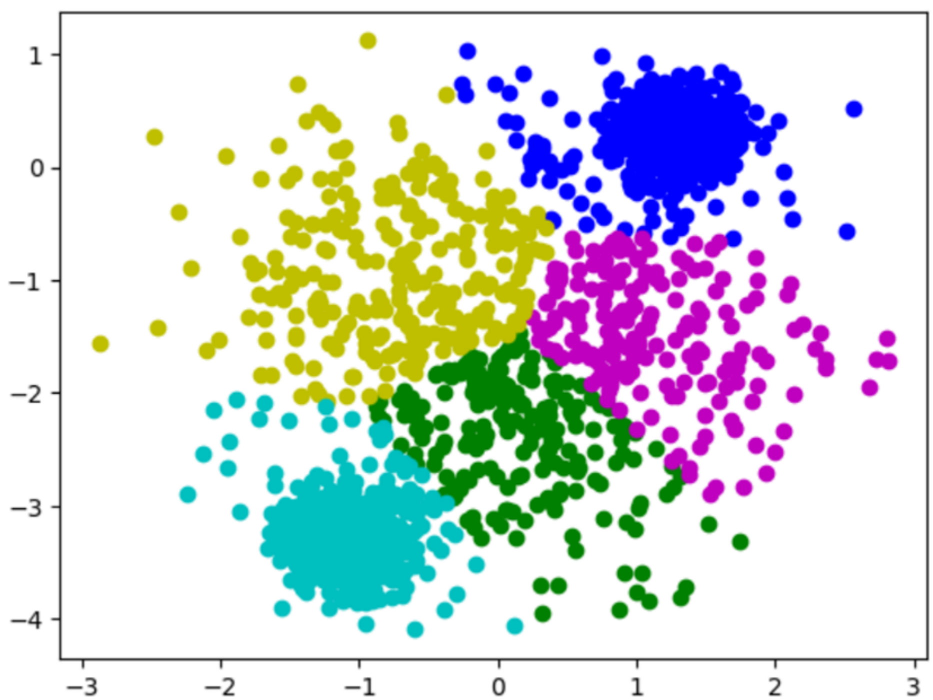


Run 2

Pytorch, score = 0.2661682790286359



Scikit, score = 0.26462592838204324



Comment: when k = 5, the Pytoch implementation results in a very different clustering across multiple runs, see Pytorch clustering run 2. The resulting score is also higher than that of run 1. This is also caused by random initialization.

Part 2: Mixture of Gaussians (MoG)
Function Explanation In Comment
Function **distanceFunc (X, MU)**

```
"""
Inputs:
X: Dataset Matrix (N By D) (N By 2) (2D)
MU: Cluster Mean Matrix (K By D) (K By 2) (2D)

Outputs:
pair_dist: Pairwise Distance Matrix (N By K)
"""
# Calculating Distance Between Each Data Point And Cluster Mean
def distanceFunc(X, MU):
 # TODO: explain this function in your report
 # Unsqueeze Dataset Matrix
 # [[[X11], [X12]], ... , [[XN1], [XN2]]]
 X1 = torch.unsqueeze(X, -1)
 # Unsqueeze Transpose Of Cluster Mean Matrix (D By K) (2 By K)
 # [[u11, ... , uN1], [u12, ... , uN2]]
 MU1 = torch.unsqueeze(MU.T, 0)

 # Calculate L2 Norm Between Each Data Point And Each Cluster Mean
 pair_dist = torch.sum((X1 - MU1) ** 2, 1)

 # Return Result
 return pair_dist
```

Function **log_GaussPDF(X, mu, sigma)**

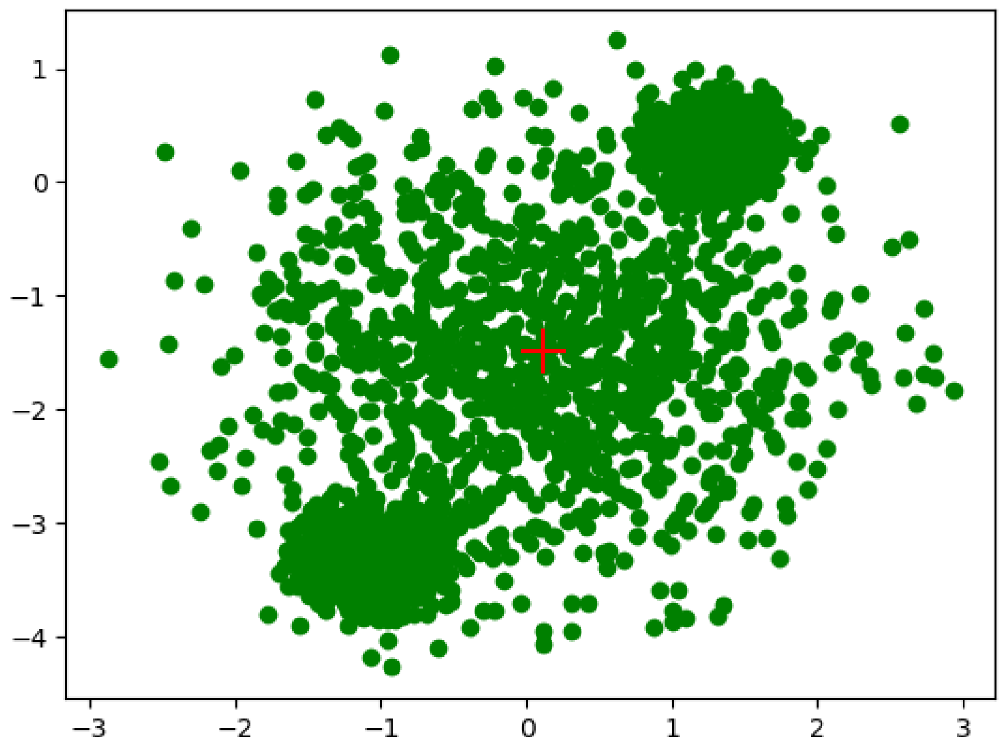
```
"""
Inputs:
X: Dataset Matrix (N By D)
MU: Cluster Means (K By D)
Sigma: Single Variance Of Each Cluster (K By 1)

Outputs:
log_PDF: Log Gaussian PDF (N By K)

"""

# Calculate log of probability density function for all clusters
def log_GaussPDF(X, mu, sigma):
 # TODO: explain this function in your report
 dim = X.shape[-1]

 Pi = torch.tensor(float(np.pi))
 sigma_2 = (torch.square(sigma)).T # 1 X K
 diff = distanceFunc(X, mu)  # N X K

 log_PDF = diff / sigma_2  # N X K
 log_PDF += dim * torch.log(2 * Pi)
 log_PDF += dim * torch.log(sigma_2)
 log_PDF *= -0.5  # N X K

 return log_PDF
```

Function **log_posterior(log_PDF, log_pi)**

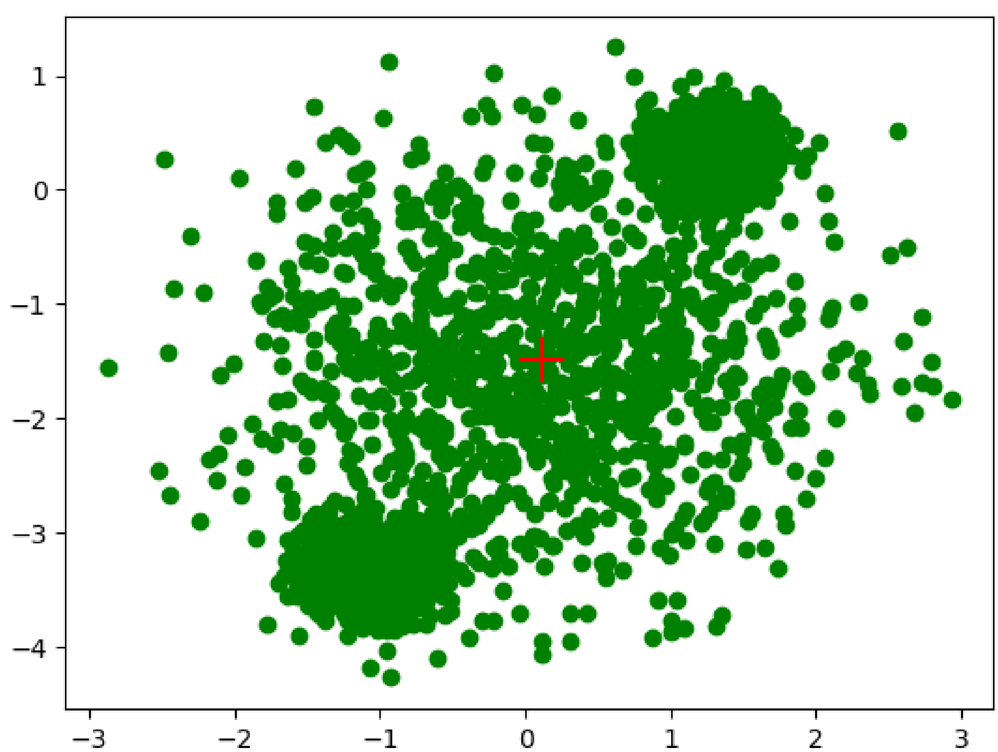
```
"""
Inputs:
log_PDF: Log Gaussian PDF (N By K)
log_pi: Prior Probability Of Clusters (K By 1)

Outputs:
log_joint: All Log Of Joint Probability Between Prior Probability And
Gaussian PDF (N By K)
log_marginal: Summation Of Log Of Joint Probability (N By 1)
"""

# Compute log probability of the cluster variable k given each data points
def log_posterior(log_PDF, log_pi):
 # TODO: Explain this function in your report
 log_joint = log_PDF + log_pi.T  # N X K
 log_marginal = torch.logsumexp(log_joint, dim = 1)  # N x 1
  return log_joint, log_marginal
```

Test GMM Function Visualization

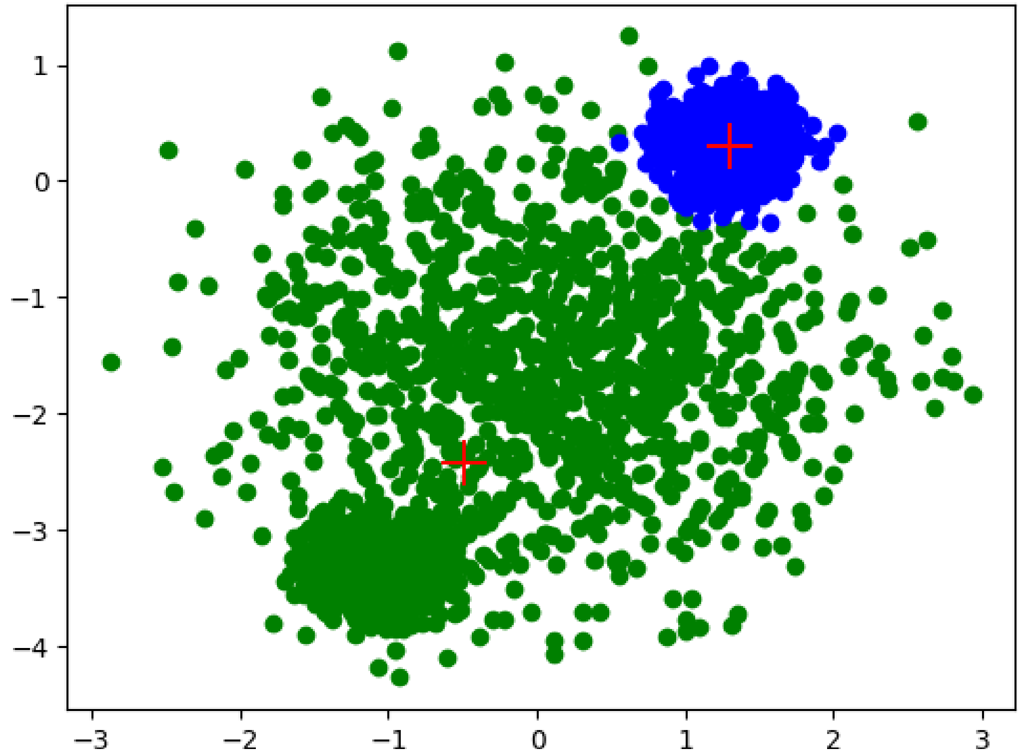
K = 1, init_kmeans = True



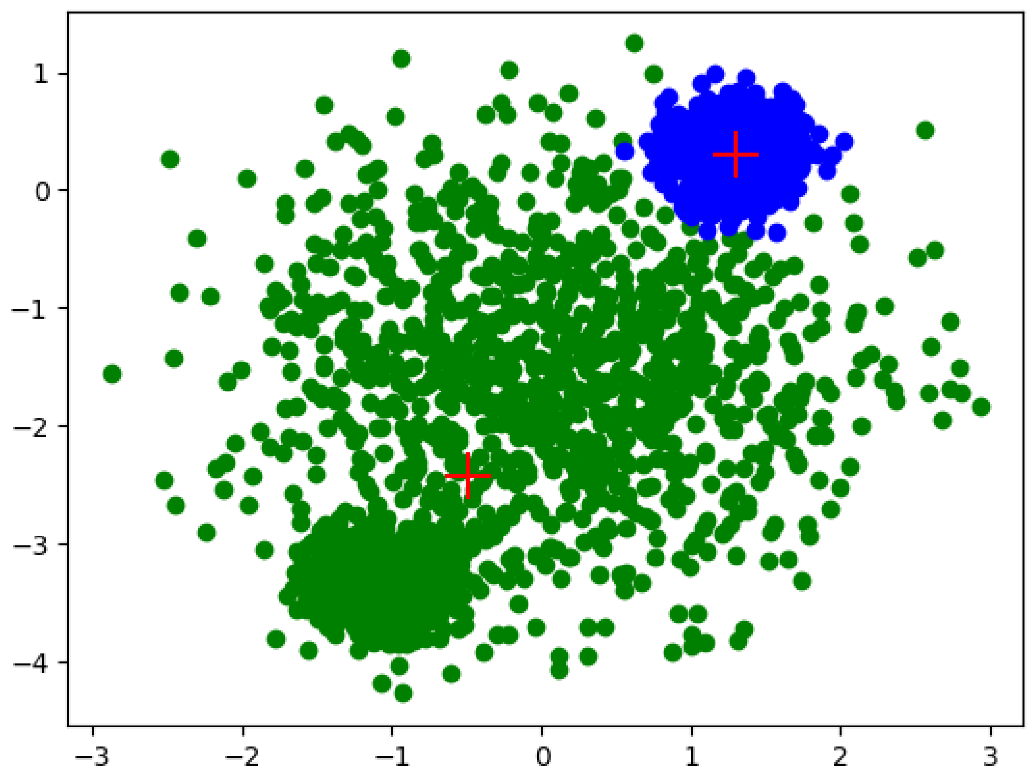
K = 1, init_kmeans = False



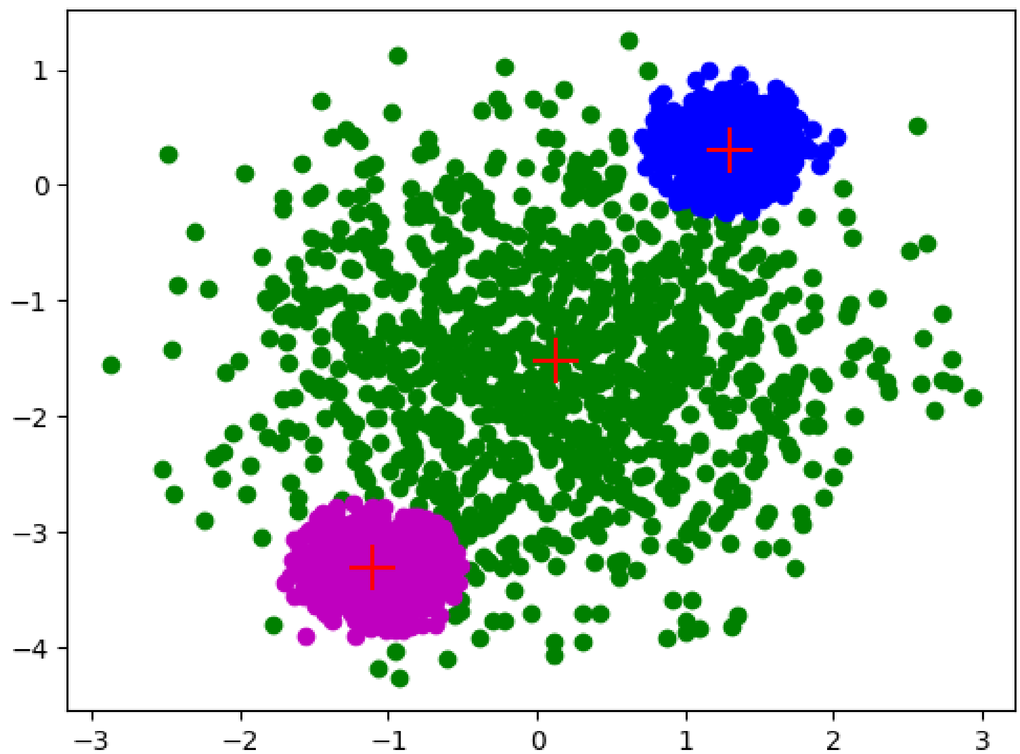
K = 2, init_kmeans = True



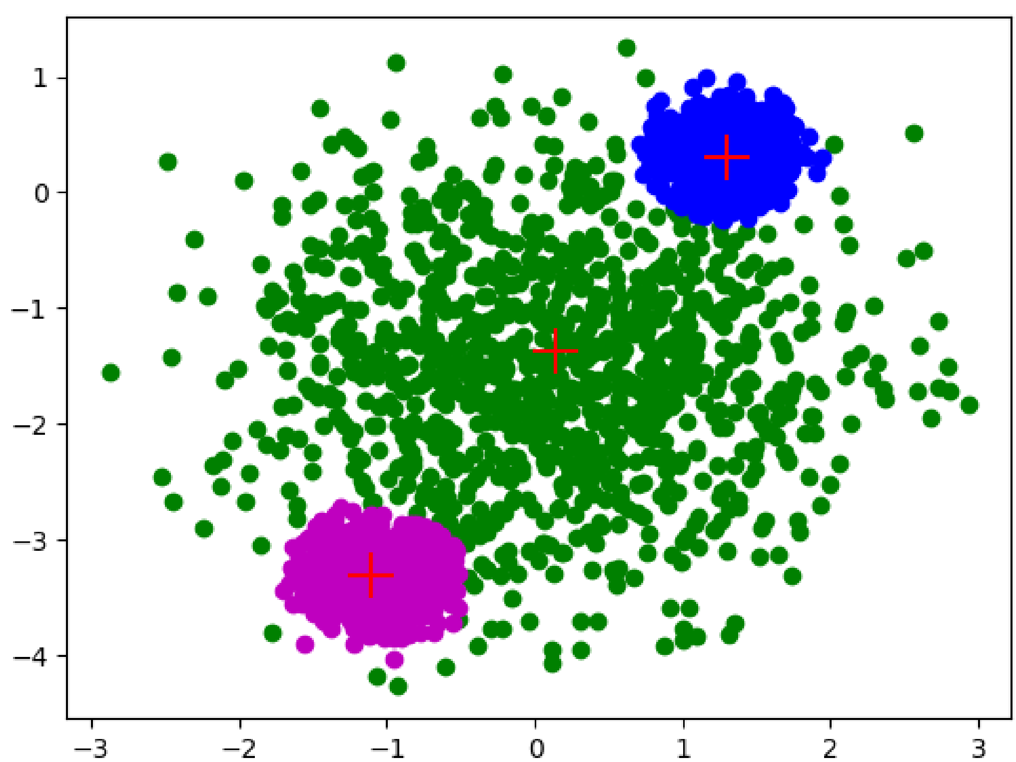
K = 2, init_kmeans = False



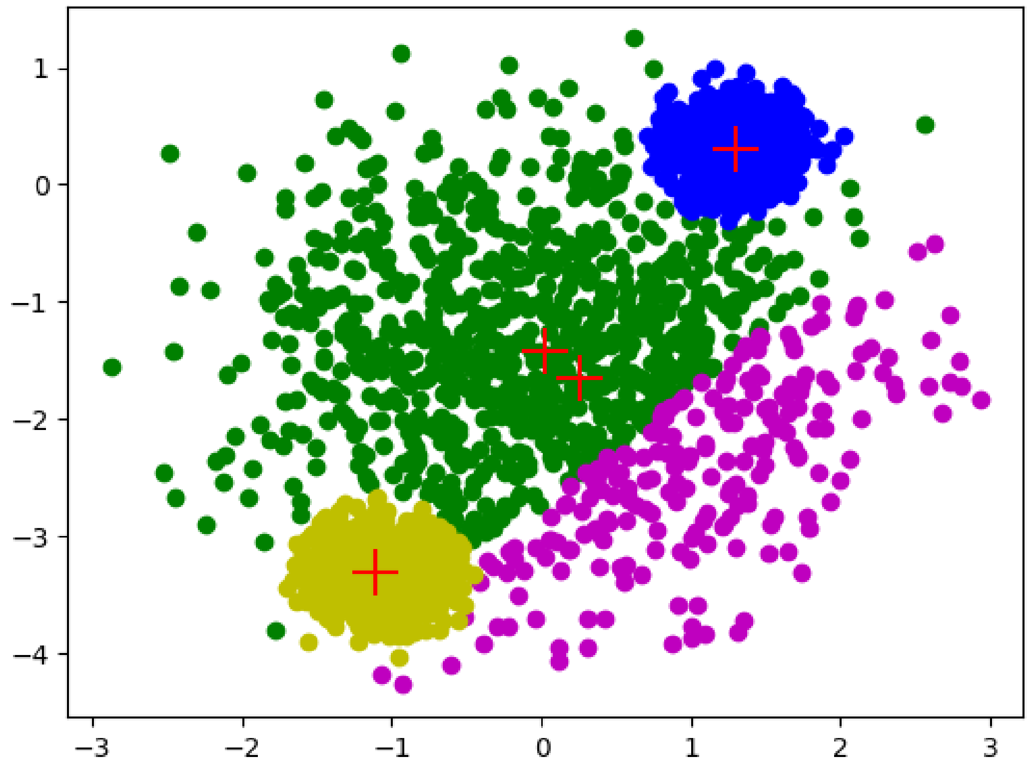
K = 3, init_kmeans = True



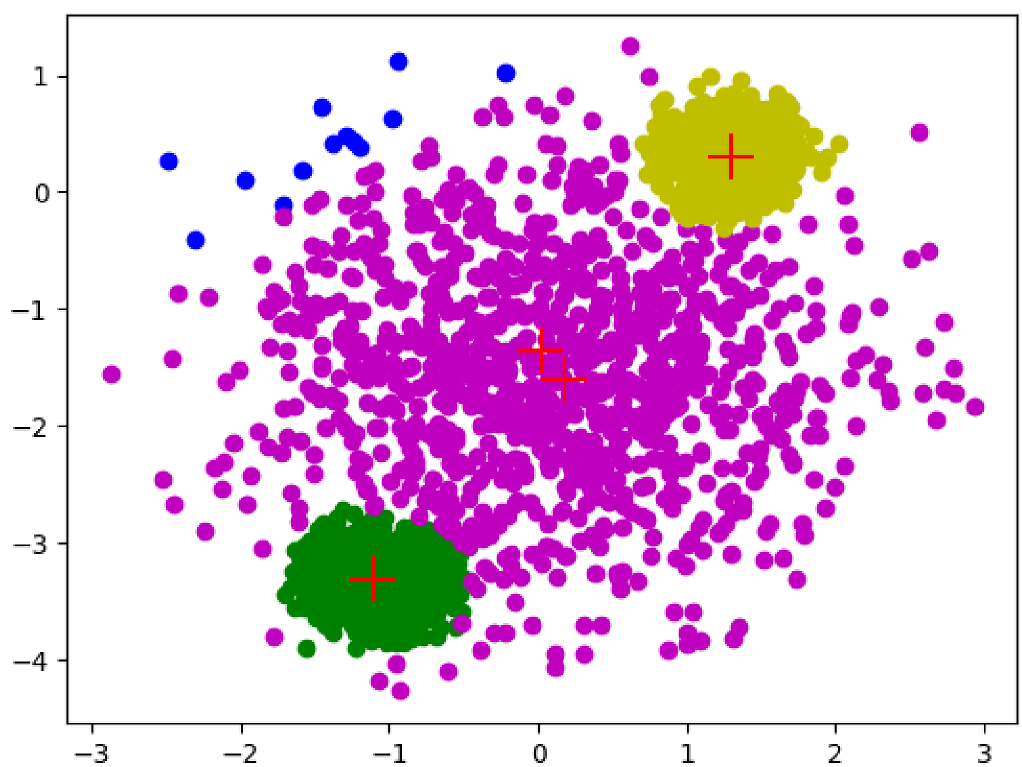
K = 3, init_kmeans = False



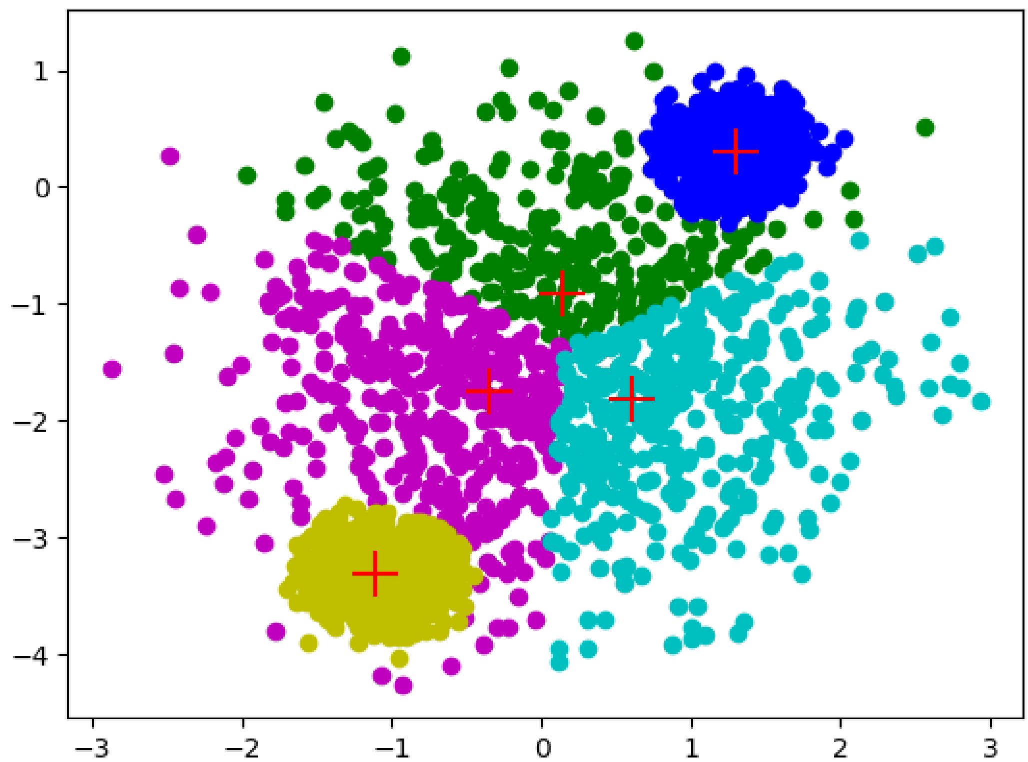
K = 4, init_kmeans = True



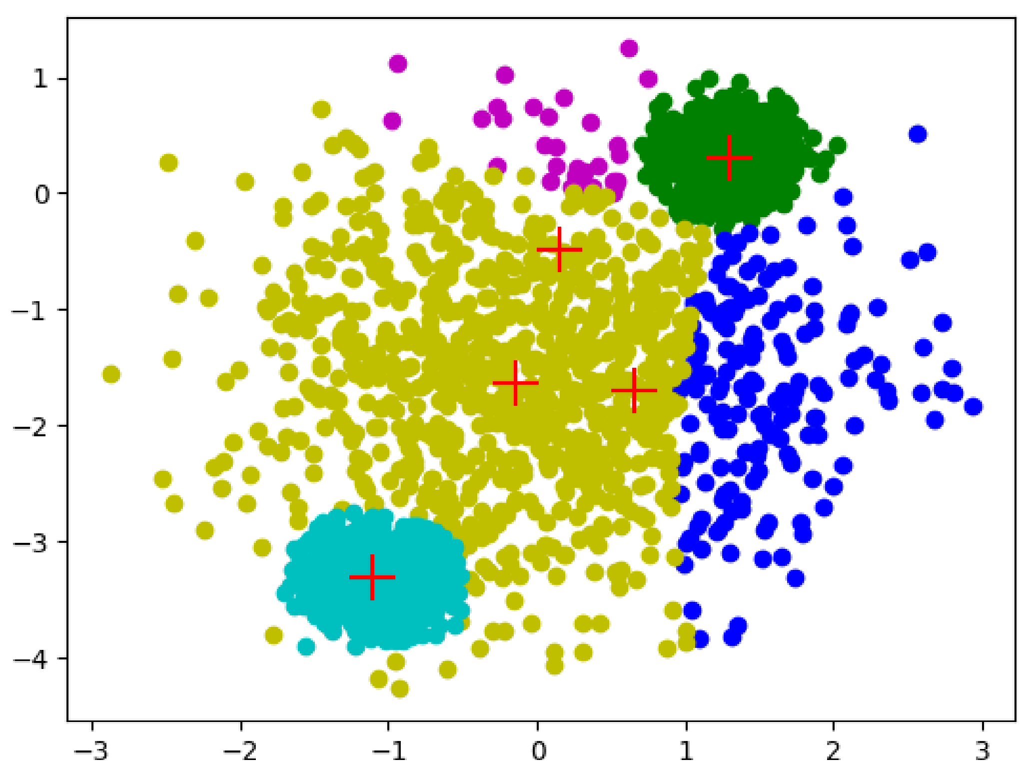
K = 4, init_kmeans = False



K = 5, init_kmeans = True



K = 5, init_kmeans = False



Comment:
When K is less than or equal to three, the clusters generated with or without K means initialized were quite similar. When K is greater than or equal to four, the clusters generated with K means initialized yielded much better distribution than K means not initialized. Thus, we have concluded that GMM with K means initialized is better.