Part 1:

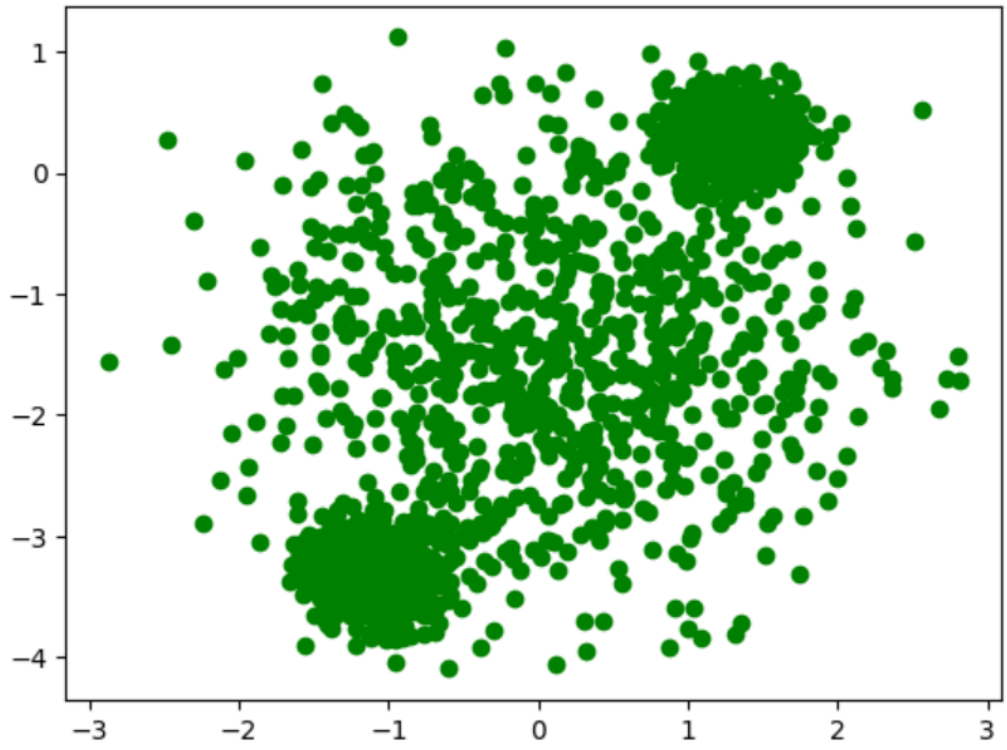
Compare the clustering performance between our PyTorch implementation and the scikit-learn one. You can do this by including in your report the visualization results between two implementations for k=1,2,3,4,5. Make sure to run them multiple times to see if the results are different for different runs. You can use test_sckitlearn and test_pytorch for this purpose. Comment your results.

For each k value, test_pytorch and test_sckitlearn ran 10 times each. Representative graphs are selected to show in the report.

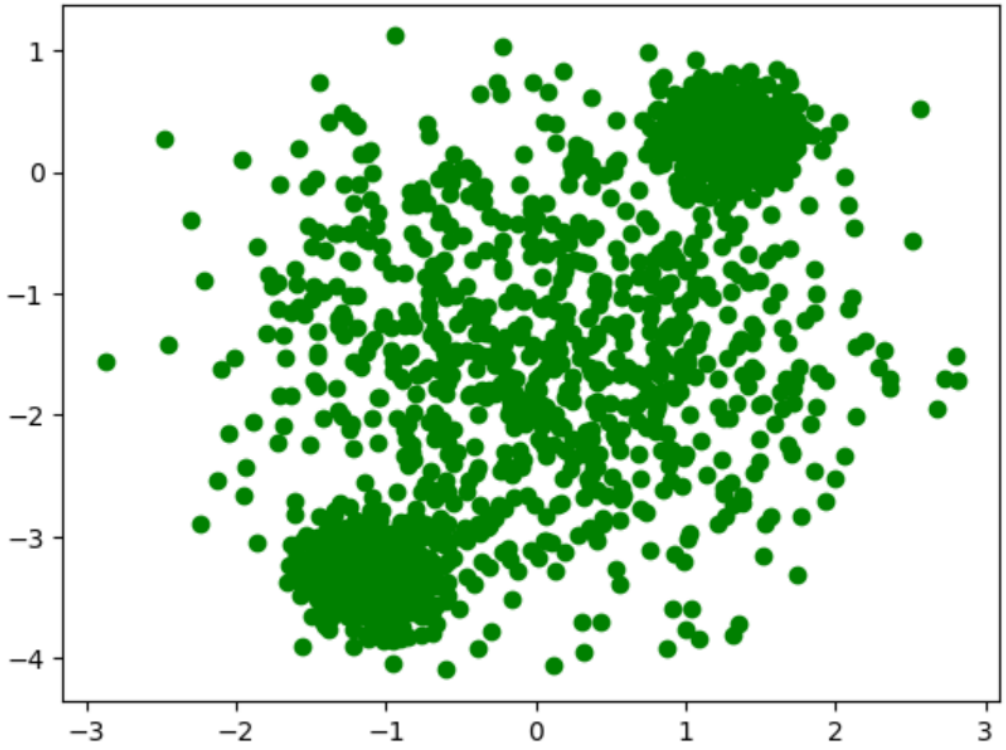
K = 1:

```
1 :
PyTorch test score: 3.8863279958945642
Scikit-learn test score: 3.8862500723240965
2 :
PyTorch test score: 3.886310718132602
Scikit-learn test score: 3.8862500723240965
3 :
PyTorch test score: 3.886152639151071
Scikit-learn test score: 3.8862500723240965
4 :
PyTorch test score: 3.886298948491628
Scikit-learn test score: 3.8862500723240965
5 :
PyTorch test score: 3.8862882915152426
Scikit-learn test score: 3.8862500723240965
6 :
PyTorch test score: 3.886204535930282
Scikit-learn test score: 3.8862500723240965
7 :
PyTorch test score: 3.886235989150467
Scikit-learn test score: 3.8862500723240965
8 :
PyTorch test score: 3.886251717274558
Scikit-learn test score: 3.8862500723240965
9 :
PyTorch test score: 3.8861896778524825
Scikit-learn test score: 3.8862500723240965
10 :
PyTorch test score: 3.886302879547527
Scikit-learn test score: 3.8862500723240965
```

Pytorch plot:



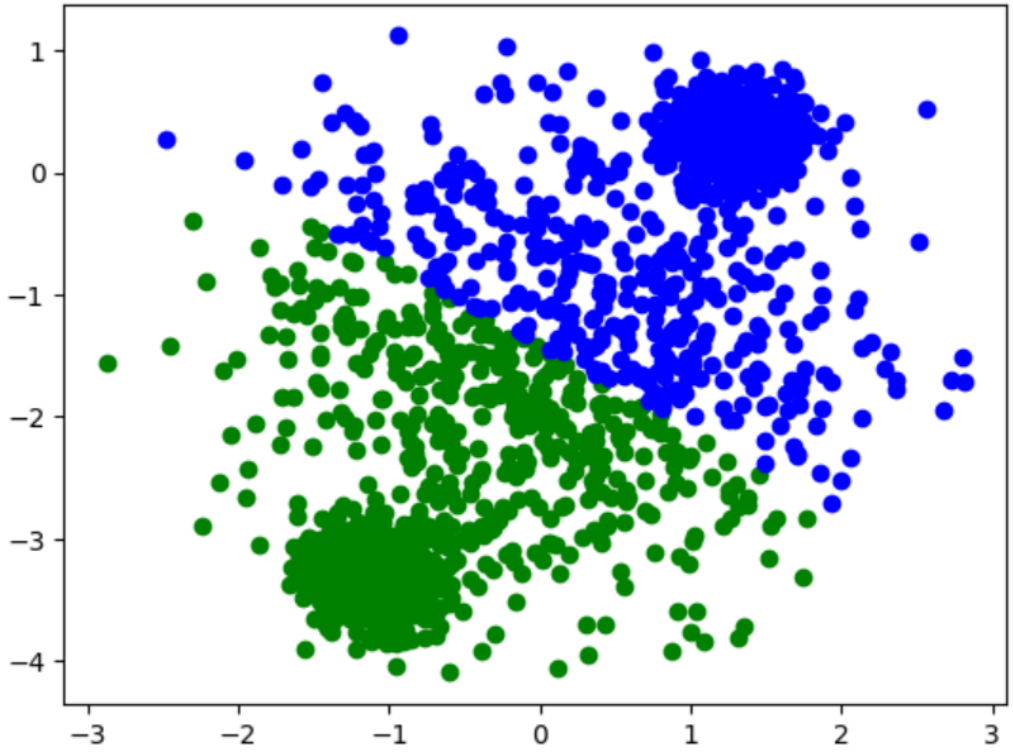
Scikit Learn plot:



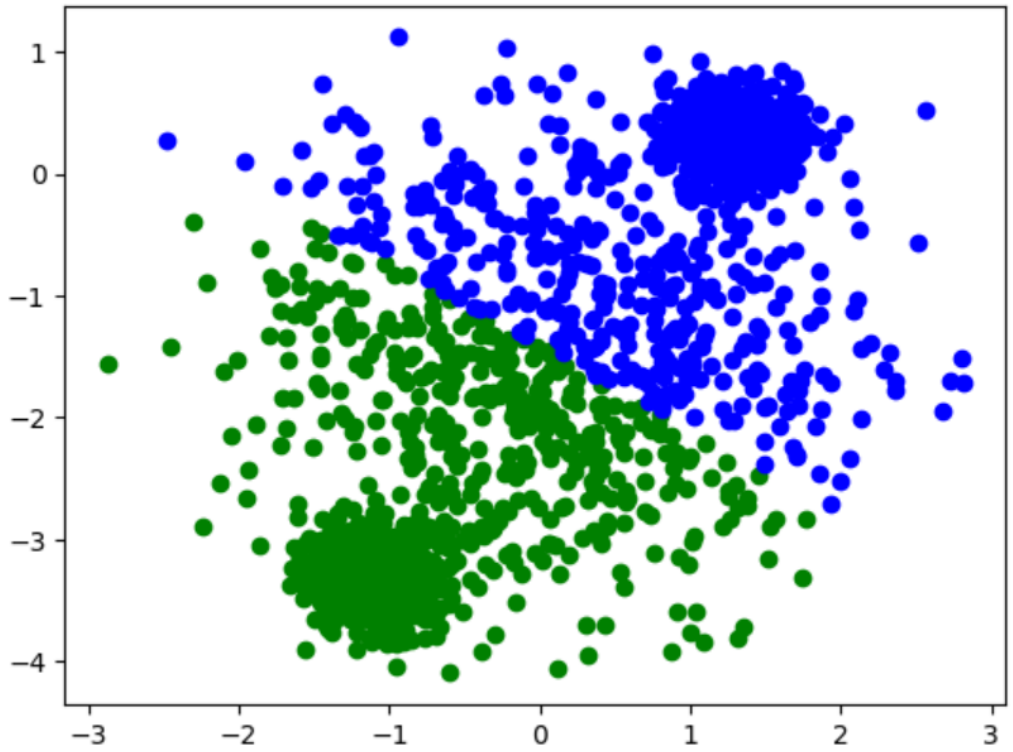
K = 2:

```
1 :
PyTorch test score: 0.8749690009021212
Scikit-learn test score: 0.8749609567839518
2 :
PyTorch test score: 0.8749483867841554
Scikit-learn test score: 0.8749609567839518
3 :
PyTorch test score: 0.8749577457690865
Scikit-learn test score: 0.8749609567839518
4 :
PyTorch test score: 0.8749046387798589
Scikit-learn test score: 0.8749609567839518
5 :
PyTorch test score: 0.8749882547486472
Scikit-learn test score: 0.8749609567839518
6 :
PyTorch test score: 0.8749436226323832
Scikit-learn test score: 0.8749609567839518
7 :
PyTorch test score: 0.8749034890894473
Scikit-learn test score: 0.8749609567839518
8 :
PyTorch test score: 0.8750319726067404
Scikit-learn test score: 0.8749609567839518
9 :
PyTorch test score: 0.8749755274835428
Scikit-learn test score: 0.8749609567839518
10 :
PyTorch test score: 0.874977417247387
Scikit-learn test score: 0.8749609567839518
```

Pytorch plot:



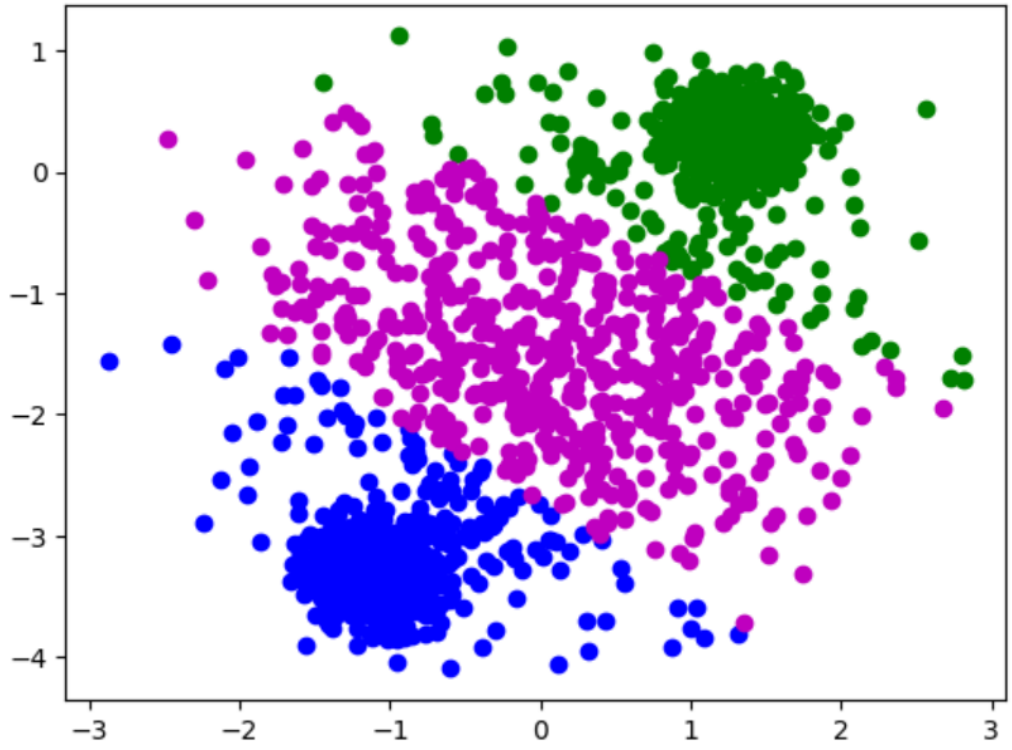
Scikit Learn plot:



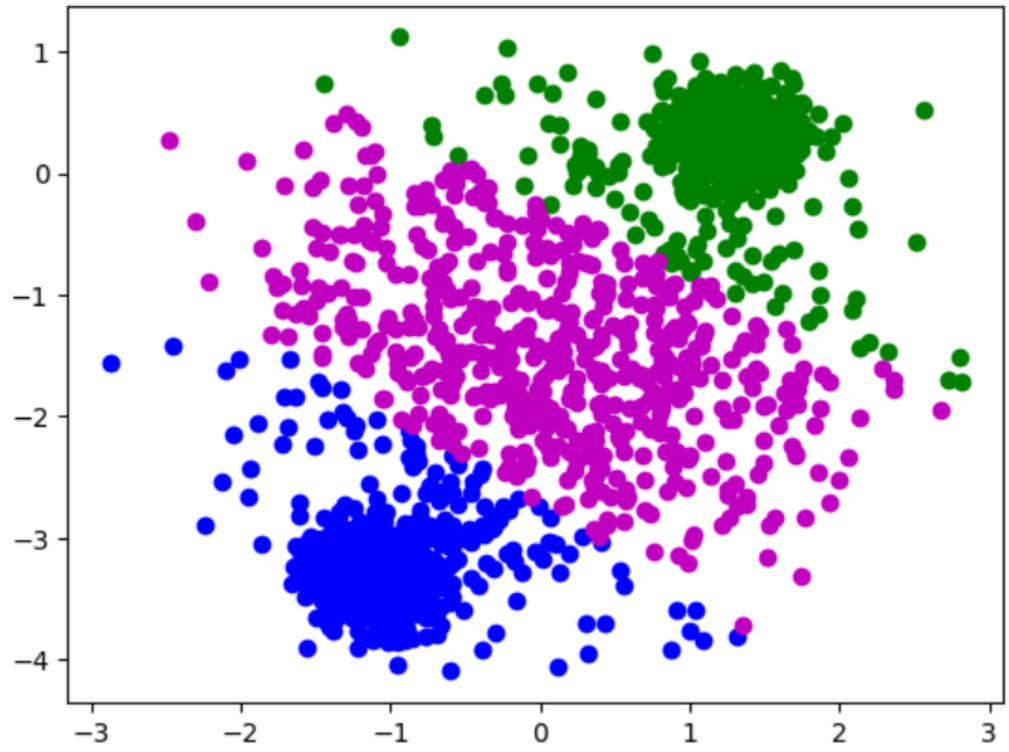
K = 3:

```
1 :
PyTorch test score: 0.4884948001147599
Scikit-learn test score: 0.488244936700977
2 :
PyTorch test score: 0.4885274272744623
Scikit-learn test score: 0.488244936700977
3 :
PyTorch test score: 0.4884674133603 8496
Scikit-learn test score: 0.488244936700977
4 :
PyTorch test score: 0.488643639734 2508
Scikit-learn test score: 0.488244936700977
5 :
PyTorch test score: 0.4884690648841 0075
Scikit-learn test score: 0.488244936700977
6 :
PyTorch test score: 0.4887083413516 7296
Scikit-learn test score: 0.488244936700977
7 :
PyTorch test score: 0.4884736615579186
Scikit-learn test score: 0.488244936700977
8 :
PyTorch test score: 0.4883875331035557
Scikit-learn test score: 0.488244936700977
9 :
PyTorch test score: 0.4885143898329381
Scikit-learn test score: 0.488244936700977
10 :
PyTorch test score: 0.4885559116653 5633
Scikit-learn test score: 0.488244936700977
```

Pytorch plot:



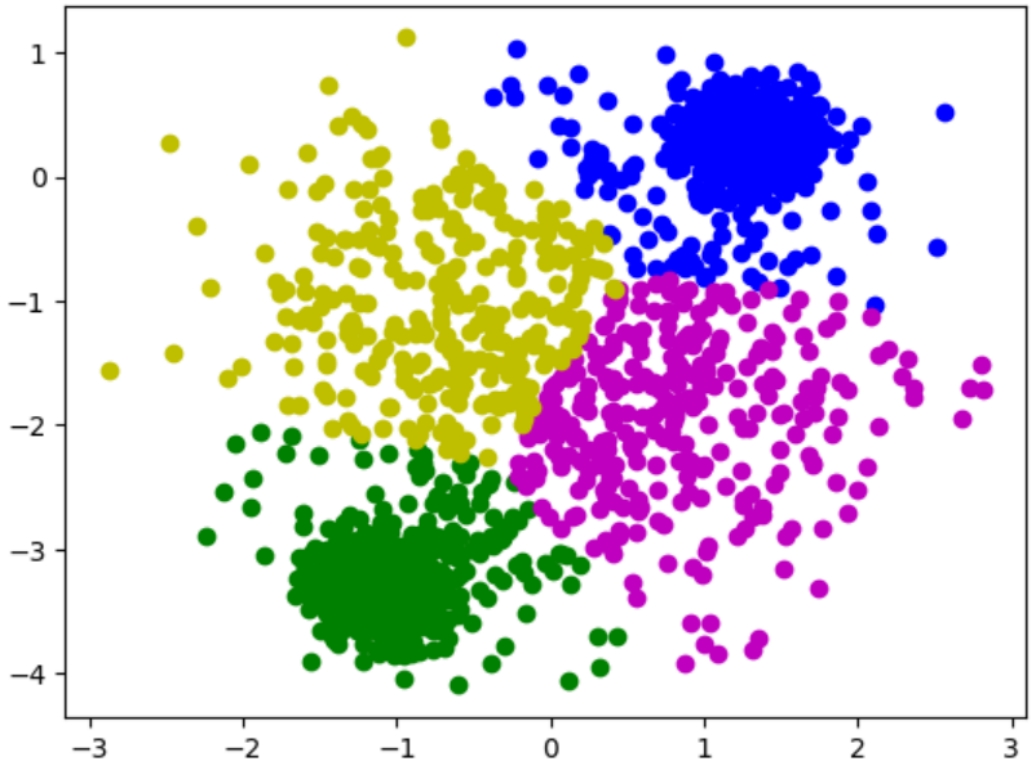
Scikit Learn plot:



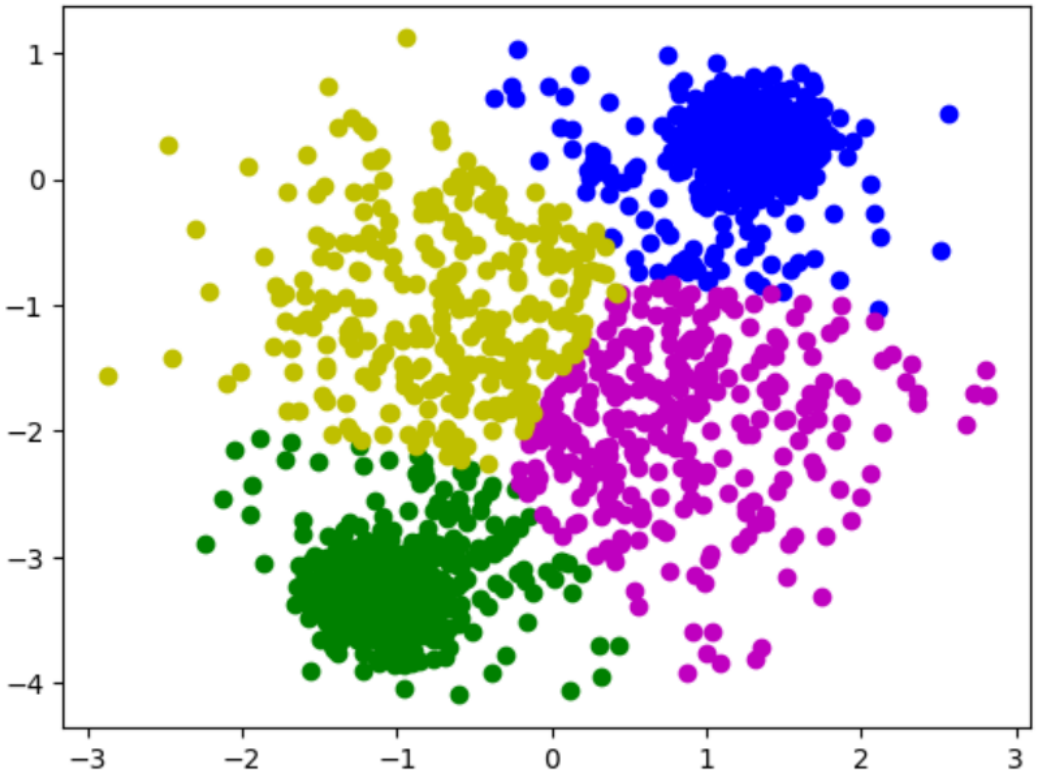
K = 4:

```
1 :
PyTorch test score: 0.3151518452108885
Scikit-learn test score: 0.3149894361501133
2 :
PyTorch test score: 0.315101126934532
Scikit-learn test score: 0.3149894361501133
3 :
PyTorch test score: 0.315138512757640 7
Scikit-learn test score: 0.3149894361501133
4 :
PyTorch test score: 0.3150069613708341
Scikit-learn test score: 0.3149894361501133
5 :
PyTorch test score: 0.315132809798558
Scikit-learn test score: 0.3149894361501133
6 :
PyTorch test score: 0.315149389528058 7
Scikit-learn test score: 0.3149894361501133
7 :
PyTorch test score: 0.315129461141735
Scikit-learn test score: 0.3149894361501133
8 :
PyTorch test score: 0.315132595868406 5
Scikit-learn test score: 0.3149894361501133
9 :
PyTorch test score: 0.3150963521177545 5
Scikit-learn test score: 0.3149894361501133
10 :
PyTorch test score: 0.315128102272506 5
Scikit-learn test score: 0.3149894361501133
```

Pytorch plot:



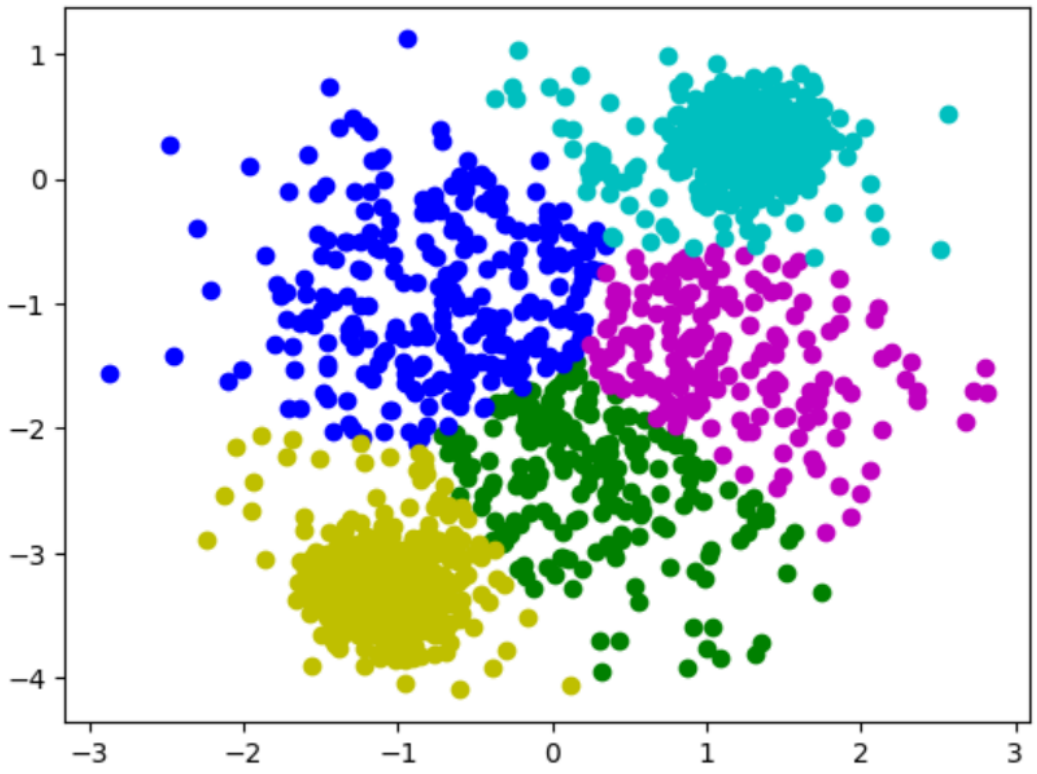
Scikit Learn plot:



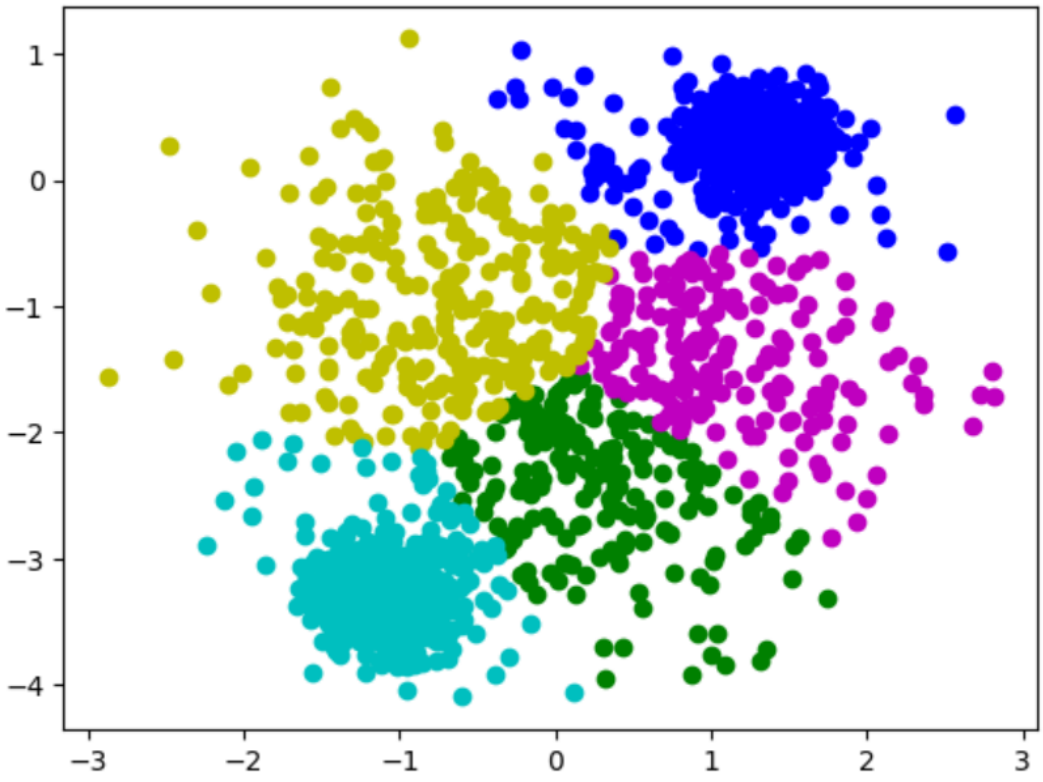
K = 5:

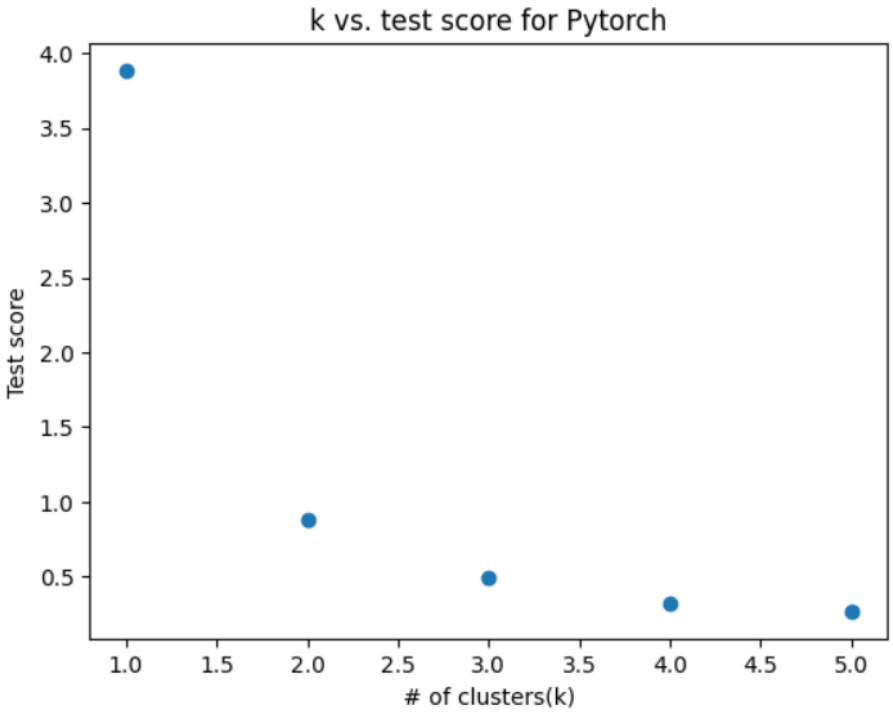
```
1 :
PyTorch test score: 0.2657750432058738 6
Scikit-learn test score: 0.2646259283820432 4
2 :
PyTorch test score: 0.2643702417485735 4
Scikit-learn test score: 0.2646259283820432 4
3 :
PyTorch test score: 0.264444463229941 3
Scikit-learn test score: 0.2646259283820432 4
4 :
PyTorch test score: 0.2747871205964730 3
Scikit-learn test score: 0.2646259283820432 4
5 :
PyTorch test score: 0.267362797219244 3
Scikit-learn test score: 0.2646259283820432 4
6 :
PyTorch test score: 0.266425845777722 1
Scikit-learn test score: 0.2646259283820432 4
7 :
PyTorch test score: 0.264375712512615 77
Scikit-learn test score: 0.2646259283820432 4
8 :
PyTorch test score: 0.264329239474609 3
Scikit-learn test score: 0.2646259283820432 4
9 :
PyTorch test score: 0.2771083141673238 4
Scikit-learn test score: 0.2646259283820432 4
10 :
PyTorch test score: 0.2643740627911213 6
Scikit-learn test score: 0.2646259283820432 4
```

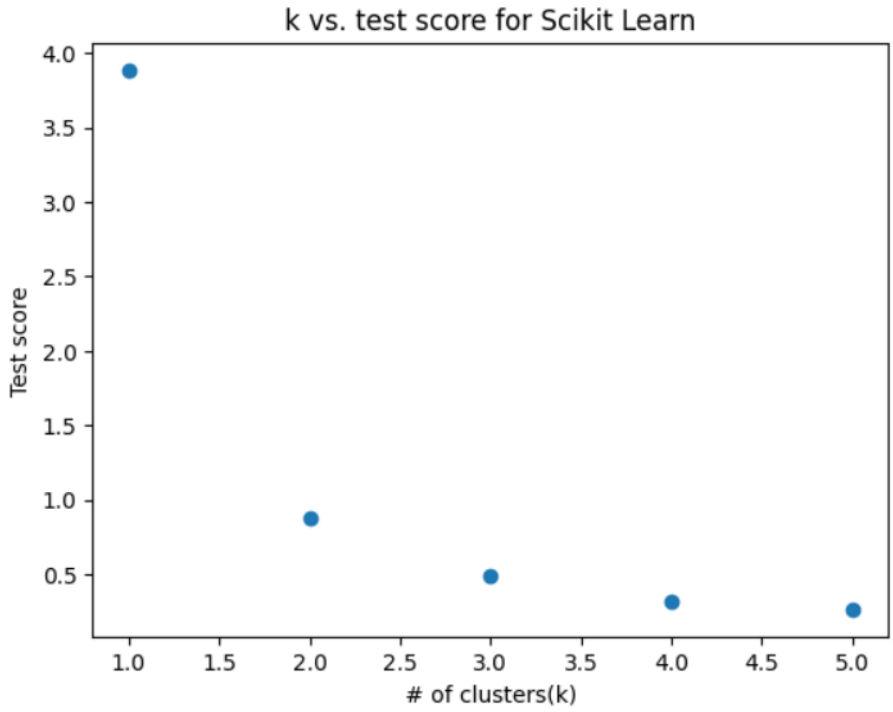
Pytorch plot:



Scikit Learn plot:







Each data point is the average of the 10 experiment scores

For the Scikit Learn Kmean, as the number of clusters increases, the test score, which measures the average distance from any point to their closest cluster center, goes down. Similar trend is seen in Pytorch Kmean, where the test scores decrease as the number of

clusters increases. Performance wise, for each k value, both Scikit-learn and Pytorch Kmean generated extremely similar and consistent scores across their 10 experiments. Visually, the two implementations generated extremely similar graphs.

The results of the Kmean algorithm makes sense. As the number of clusters increases, the average distance from any datapoint to its closest cluster center decreases, because the algorithm is able to create more defined clusters that can better capture the structure of the data. This means that each data point is more likely to be assigned to a cluster that closely matches its own characteristics, resulting in a smaller distance between the data point and its assigned cluster center.

Part 2:

Explain the functionality of these functions in your report (every line).

1. distanceFunc(X, MU): what are X and MU? What is the purpose of this function? Explains the output.

   X and MU are tensors where X is the input data with shape (# of samples, # of features) and MU is the cluster centers with shape (# of clusters, # of features). The purpose of this function is to calculate the Euclidean distance between each point in the input data X and each cluster center in MU.

   ```
   X1 = torch.unsqueeze(X, -1)
   MU1 = torch.unsqueeze(MU.T, 0)
   ```

   The first line inserts a new dimension of size one to X at the end (axis = -1 -> last dimension), the updated X is stored in the variable X1. The second line first transposes MU and then inserts a new dimension of size one to MU transposed at the beginning (axis = 0 -> first dimension), the updated MU is stored in the variable MU1. The two lines of “unsqueeze” are used to reshape the input tensors X and MU to make them compatible for broadcasting in the subsequent computation.

   ```
   pair_dist = torch.sum((X1 - MU1)**2, 1)
   ```

   The third line uses the broadcasting technique to calculate the pairwise squared distance between each point and each cluster center. The pairwise distances are stored in the tensor pair_dist, which is of shape (# of samples, # of clusters).

   ```
   return pair_dist
   ```

   The last line outputs the pair_dist tensor, which is of shape (# of samples, # of clusters) containing the distance between each point and each cluster center.

2. log GaussPDF(X, mu, sigma): what are X and mu and sigma? What is the purpose of this function? Explains the output.

   X, mu, and sigma are tensors with X being the data set having shape (# of samples, # of features), mu being the gaussian means having shape (# of clusters, # of features), and sigma being the standard deviations having shape (n_clusters, 1). The purpose of this function is to calculate the log probability density function of a multivariate Gaussian distribution with mean mu and covariance sigma squared, evaluated at the input data X.

```
dim = X.shape[-1]
Pi = torch.tensor(float(np.pi))
sigma_2 = (torch.square(sigma)).T
diff = distanceFunc(X, mu)
```

The first line gets the dimension of the multivariate gaussian and stores it in the variable dim. The second line gets the value of pi and stores it in the variable Pi. The third line calculates variance by squaring sigma, then takes the transpose of it and stores it in the variable sigma_2. The fourth line calculates the pairwise distance between each point and each cluster center using distanceFunc(X, MU) and stores the tensor in the variable diff.

Log PDF of multivariate Gaussian is of the form:

$-0.5 * ((\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) + d * \log(2\pi) + |\Sigma|)$

```
log_PDF = diff / sigma_2
log_PDF += dim * torch.log(2 * Pi)
log_PDF += dim * torch.log(sigma_2)
log_PDF *= -0.5
```

The fifth to eighth line uses those values to calculate the log PDF of the multivariate Gaussian distribution. The fifth, sixth, seventh, and eighth line is responsible for the red part, the orange part, the purple part, and the blue part of the log PDF respectively. The result is a tensor of shape (# of samples, # of clusters) and is stored in the variable log_PDF.

```
return log_PDF
```

The last line outputs log_PDF, which is of shape (# of samples, # of clusters) containing the log PDF of each point evaluated at each cluster center.

3. log posterior(log PDF, log pi): what are log PDF and log pi? What is the purpose of this function? Explains the output.

   log_PDF is a tensor of shape (# of samples, # of clusters) containing the log PDF of each point evaluated at each cluster center, and log_pi is a tensor of shape (# of clusters, 1) containing the log prior probabilities of each cluster. The purpose of this function is to calculate the log posterior probability of each point in the input data X belonging to each cluster.

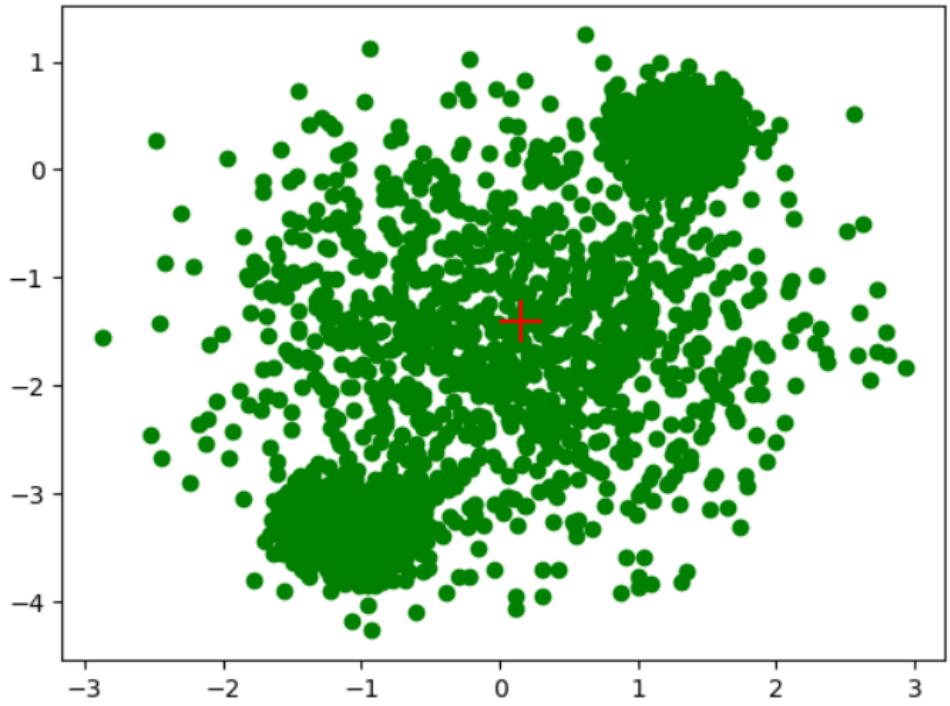
```
log_joint = log_PDF + log_pi.T
```
The first line calculates the log joint probability of each point belonging to each cluster by adding the log PDF and the log prior probability of each cluster and the log joint probability is stored in the variable log_joint, which is a tensor of shape (# of samples, # of clusters). Transpose of log_pi is used to match the dimension.

```
log_marginal = torch.logsumexp(log_joint,dim=1)
```
The second line obtains the log marginal probability of each point by summing over their joint probabilities for each cluster k and taking log. Exponential is applied to the joint probability before the summation as they are log values and Log is applied to the sum to get the log value. The log marginal probabilities are stored in the variable log_marginal, which is a tensor of shape (n_samples,).

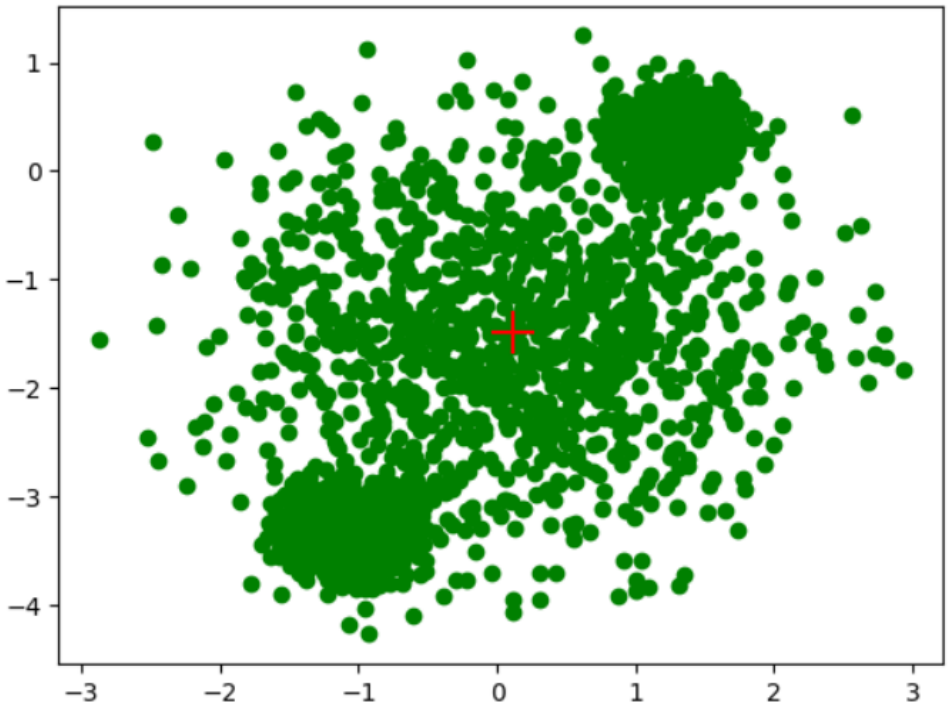
```
return log_joint, log_marginal
```
The last line outputs two tensors: log_joint containing the log joint probability of each point belonging to each cluster and is of shape (# of samples, # of clusters), and log_marginal containing the log marginal probability of each point and is of shape (n_samples,).

For each k value, test_GMM ran 10 times each for not initializing Kmean and initializing Kmean. Representative graphs are selected to show in the report.

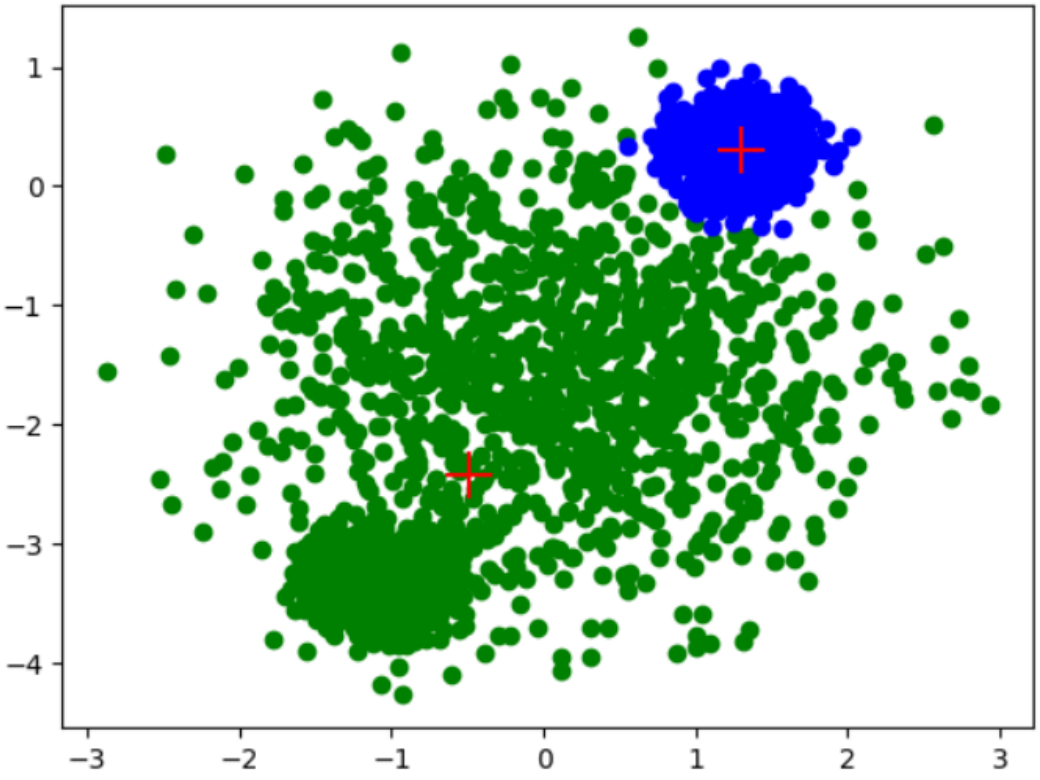
K = 1, init_kmean = False:



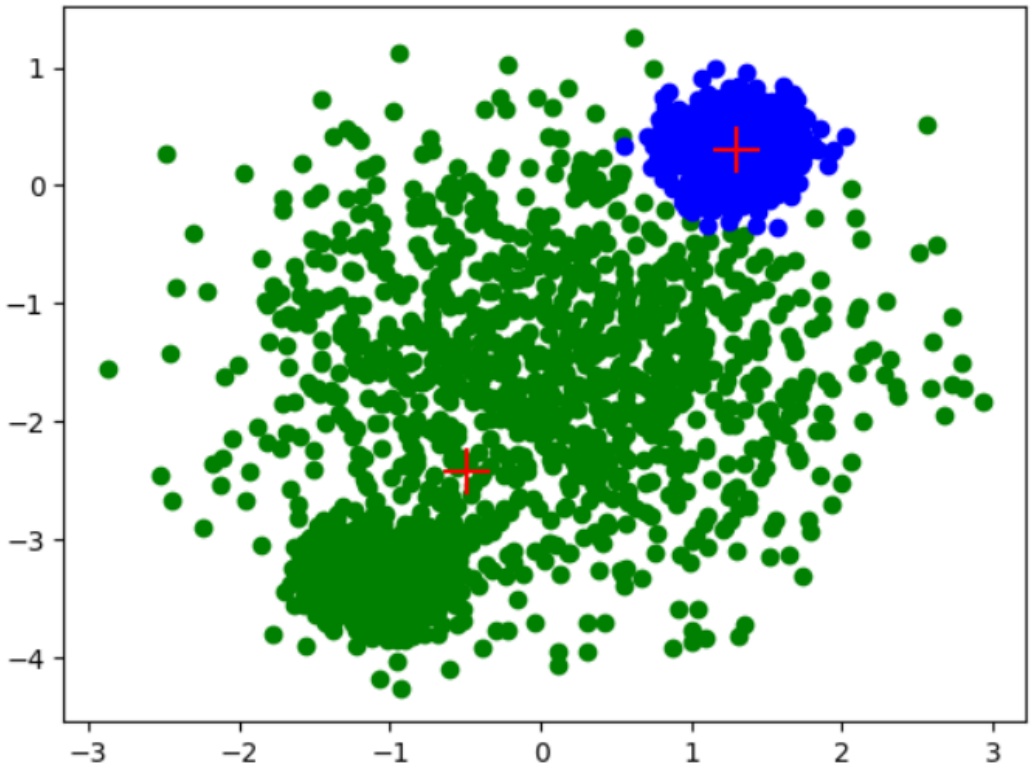
K = 1, init_kmean = True:



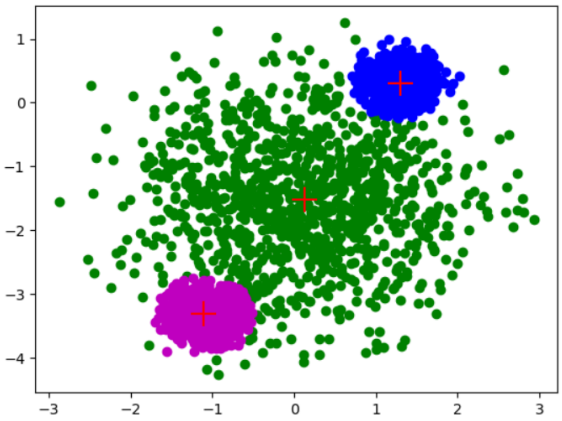
K = 2, init_kmean = False:

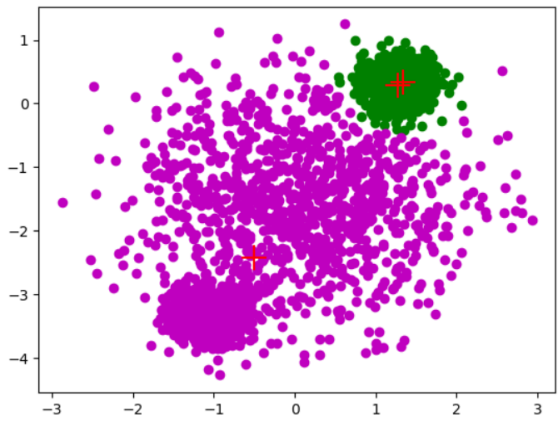


K = 2, init_kmean = True:

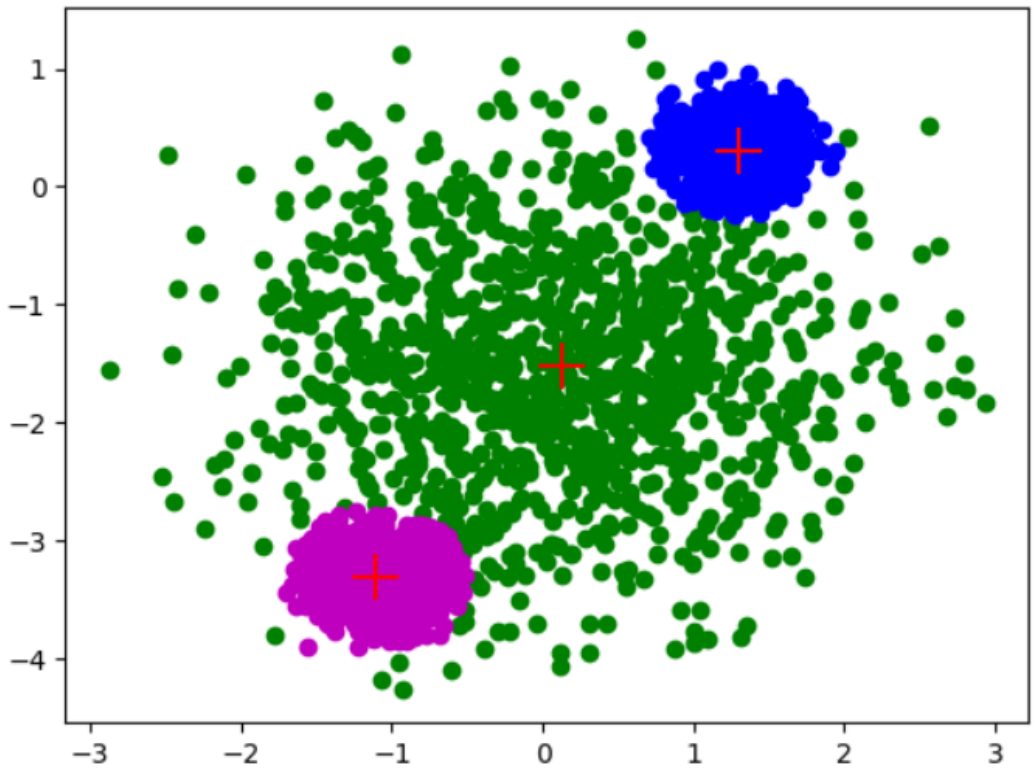


K = 3, init_kmean = False:

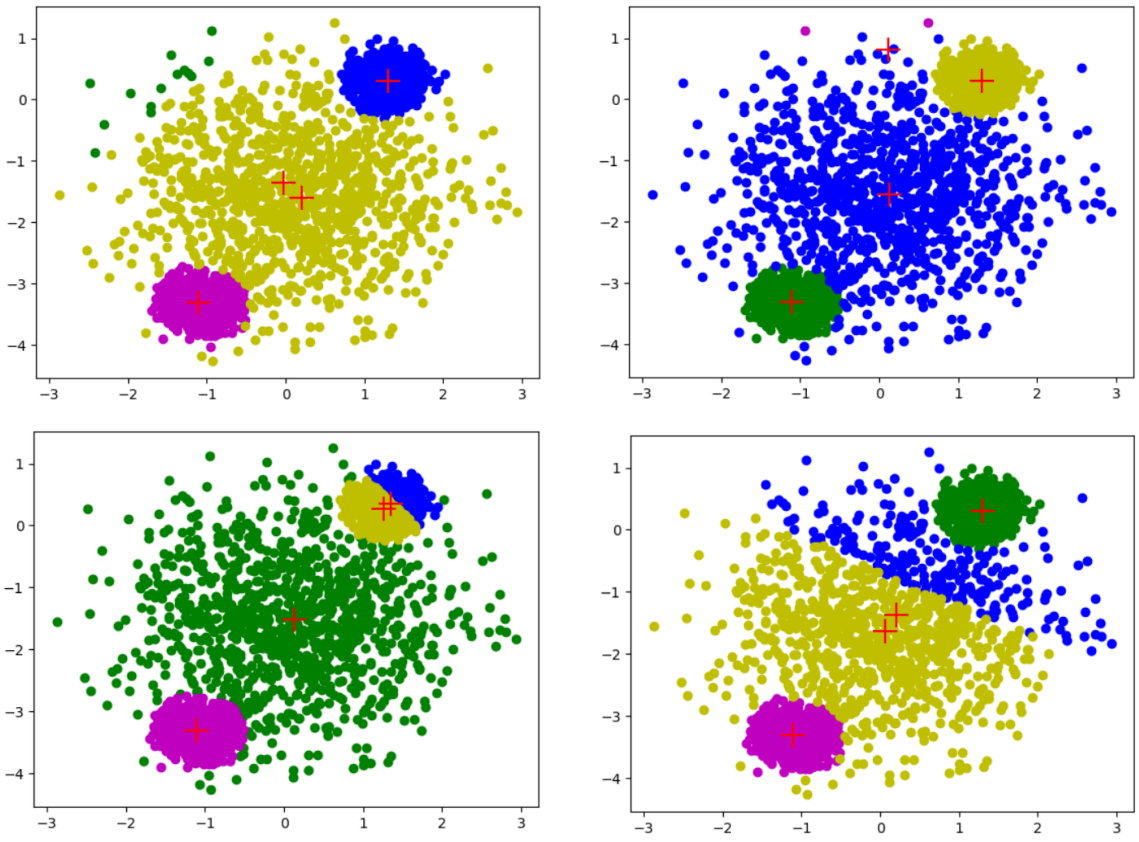




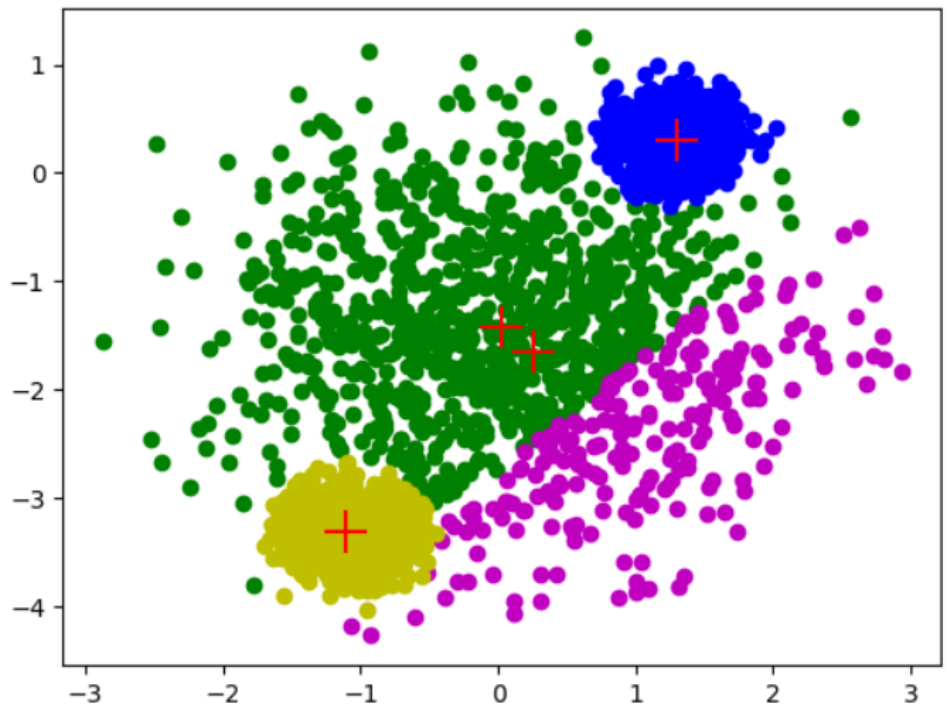
K = 3, init_kmean = True:



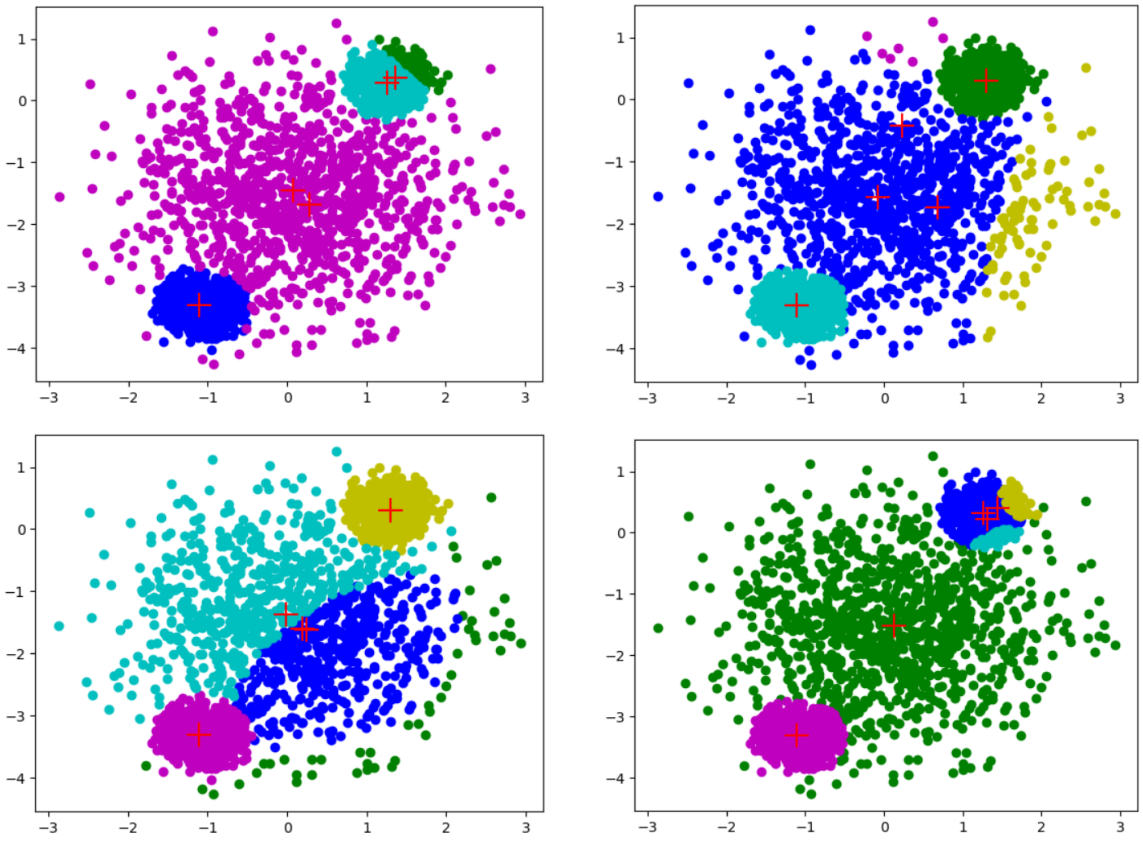
K = 4, init_kmean = False:



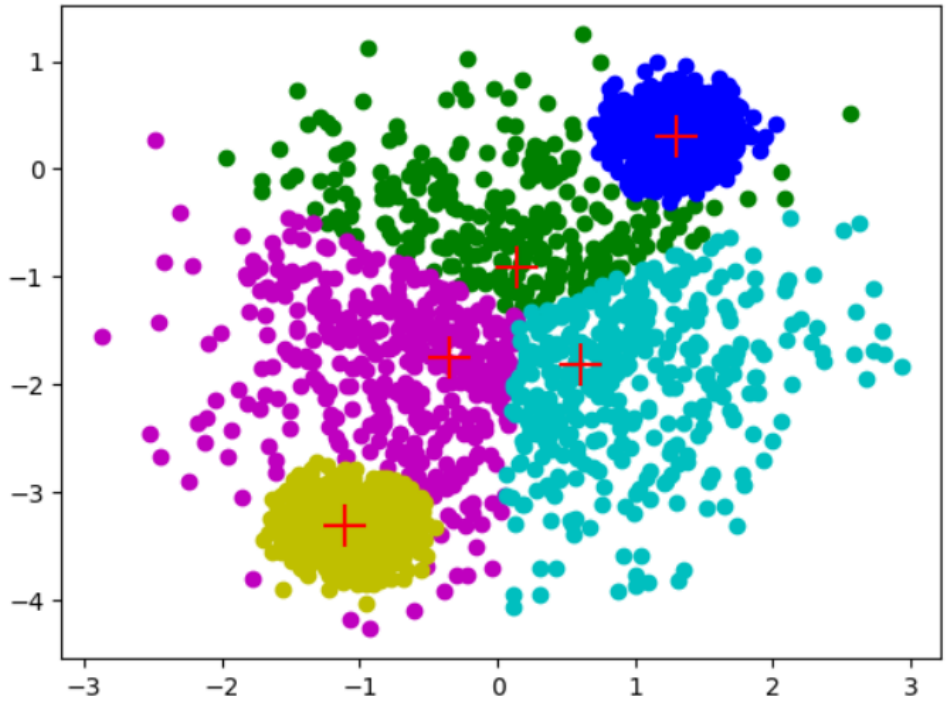
K = 4, init_kmean = True:



K = 5, init_kmean = False:



K = 5, init_kmean = True:



For test_GMM with Kmean initialized, for each k value, extremely similar cluster distributions resulted across 10 experiments, and the boundaries between clusters are consistent and stable.

For test_GMM with Kmean uninitialized, as k gets larger, the boundaries between clusters become inconsistent and unstable, varying significantly between the 10 experiments for a fixed k value.

Comparing the graphs:

- k = 1 or 2: init_kmean = False and init_kmean = True always yield extremely similar cluster distributions
- k = 3: init_kmean = False and init_kmean = True mostly yield extremely similar graphs, occasionally init_kmean = False generates unreasonable cluster distributions
- k = 4 or 5: init_kmean = False and init_kmean = True always yield different cluster distributions, where for init_kmean = False, the boundaries between clusters are inconsistent and unstable, varying significantly between experiments

By initializing Kmean first, GMM training starts with a starting set of gaussian means resulting from cluster means of the Kmean algorithm, which is much better than assigning random numbers as the starting gaussian means. This results in better convergence under limited epochs (1000 for this lab), which explains the consistency of kmean initialized test_GMM results.