Programming Assignment 2

k-means and GMM Clustering

Patrick Humphries (7097-1087-72, pvhumphr@usc.edu)

INF552 Machine Learning for Data Science (32458)

Viterbi School of Engineering

University of Southern California

Spring 2020

Abstract The object of this assignment is to familiarize the student with k-means and Gaussian Mixture Model clustering. This is accomplished in two sections, each with three parts.

Section K: k-means

- Part 1: Implementation
- Part 2: Software Familiarization
- Part 3: Applications

Section G: Gaussian Mixture Model

- Part 1: Implementation
- Part 2: Software Familiarization
- Part 3: Applications

The implementation and software familiarization for k-means was relatively quick. The scope of the implementation was merely 150 data points that are assigned to three clusters. The results of the implementation and the library solution were very similar.

The same cannot be said for GMM. Without having the professor actually solve the equations, there were no examples of how to code the equations. Experiencing negative determinants, an assumption was made that the absolute value of $|\Sigma_c|$ would be used.

The results are as follows:

data points	blue	red	green
k-means implementation	83	29	38
k-means sklearn k-means	85	31	34
GMM implementation	83	29	38
GMM sklearn mixture	86	30	34

As shown, the results are not the same, but are similar. The most difference was with the implementation versions. The difference could be attributed to rounding, limiting the number of iterations, or maybe the underlying computer language (Python versus C).

Section K: k-means

Part 1: Implementation

The Python programs "inf552_assignment_2_humphries_k_means.py" and "inf552_assignment_2_humphries_gmm.py" were developed with Jupyter Notebooks of the same name.

k-means: Algorithm

The algorithm for this assignment is relatively simple:

- Step 1: Retrieve data points.
- Step 2: Create the arbitrary number of clusters at random locations.
- Step 3: Assign each data point to the nearest cluster.
- Step 4: Compute the centroid for the cluster using the assigned data points.
- Step 5: Repeat "Step 3" and "Step 4" until each centroid converges to single location or the limit to the number of iterations is exceeded.

k-means: X Structure

The X variable is a list of data point tuples. Each tuple is the x-y coordinate. The provided contained 150 data points. The file was renamed from "clusters.txt" to "culsters.csv" so it could be inspected by a spreadsheet application. Each row of the file was a list of float values. Each row was converted into a tuple and appended to X.

```
(-1.861331241, -2.991682765)
(-2.17009237, -3.292317782)
(-1.014080969, 0.38579499)
(-2.912942536, -2.579539167)
(0.035720735, -0.799697919)

Number of data points: 150
```

k-means: C Structure

The C variable is a list of clusters. A cluster has the following attributes:

- "xy": This is the x-y tuple for the location of the centroid. As a tuple, it is easily consumed by matplotlib.
- "color": The cluster is represented by a circle of color stored in this attribute. This color is also used to visually identify all the data points assigned to the cluster.
- "X": This is a list of tuples for the data points assigned to this cluster. This list is used for calculating the centroid and displaying datapoints. Being tuples, they are easily consumed by matplotlib.

```
{'xy': (-8, 8), 'color': 'red', 'X': []}
{'xy': (-8, -8), 'color': 'blue', 'X': []}
{'xy': (8, -8), 'color': 'green', 'X': []}

Number of clusters created: 3
```

k-means: Program Structure

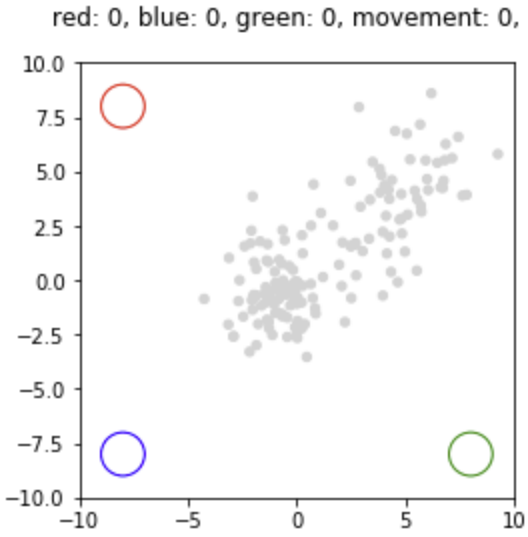
All activity in the program are carried out by functions except for the main sequence. This allowed for incremental development as well as easy understanding of the code by using meaningful function names.

```
# Main routine.
# Plot the intial state and subsequent states of clusters.
# Stop when there is no further movement of centroids.
# X:  List of data point tuples (x,y).
# C:  List of clusters.

# Load data points, create clusters, and plot the initial state.
X,C,T = initialization()

# Reassign data points to clusters, move centroids, and plot the new state.
for i in range(0,10):

    total_movement = realign_data_points(X,C,T)

    # Stop if there is relatively no movement of centroids.
    if total_movement < 0.01:
        break

# Plot using scikit library.
call_library()
```

k-means: Program Execution

Iteration 1

The data points are plotted in a grey color to indicate they yet to belong to a cluster. The empty clusters are circles with identifying colors. The clusters were placed as far away as possible from each other and the data points. This was done to emphasize the movement of the clusters and to avoid any bias.

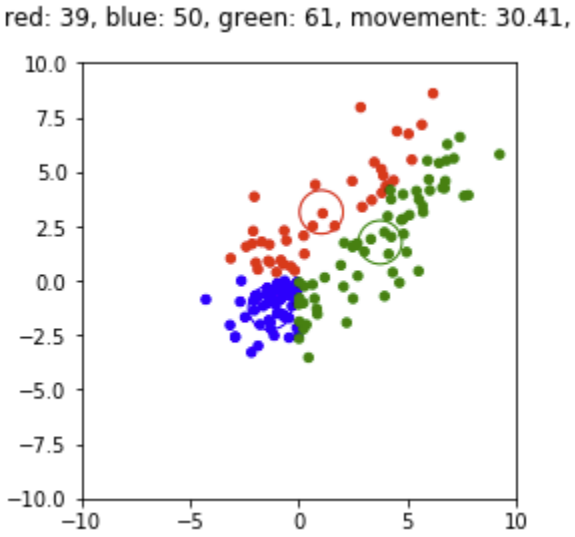


Iteration 2

I apologize for the number of iterations. However, the assignment specified "report the centroid of each cluster", so please be patient.

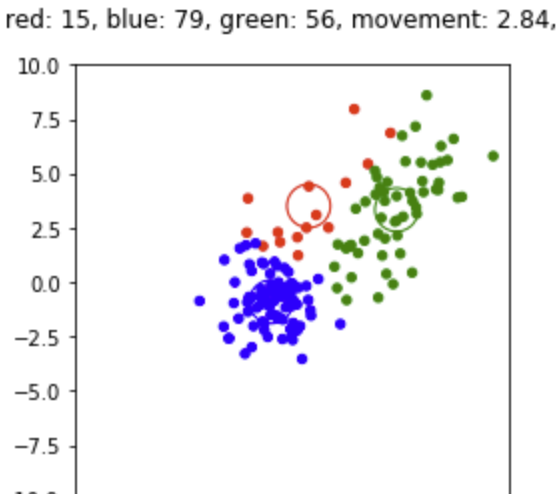
This iteration had the most movement of the centroids. The "movement" is the sum of distances moved by centroids. When this value is less than 0.01, then the iterations will stop.

As displayed, the data points were assigned to the nearest data point. Also, the new centroids will cause a significant redistribution of data points.



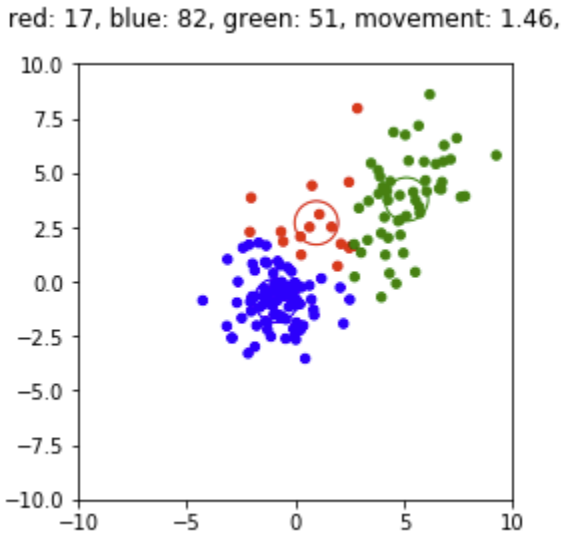
Iteration 3

As expected, there was significant reallocation of data points. The blue cluster appears to be well centered. However, the red and green clusters begin their dance.

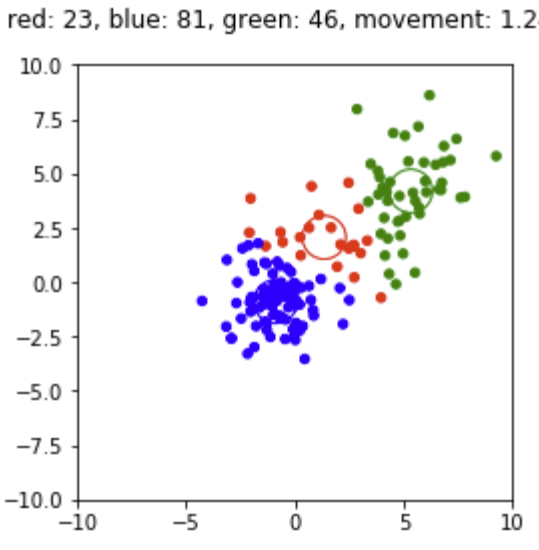


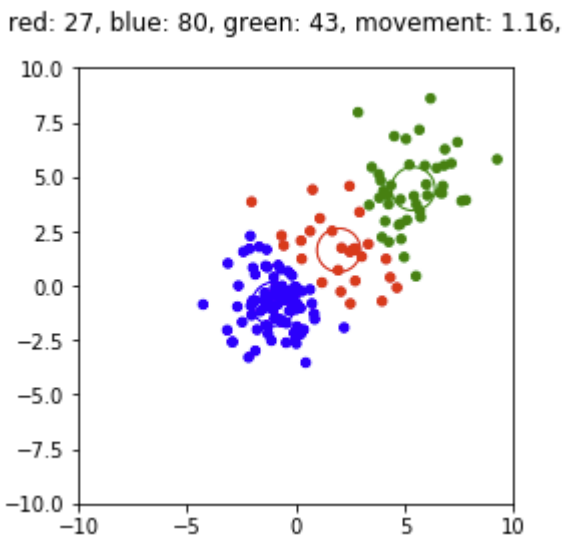
Iteration 4

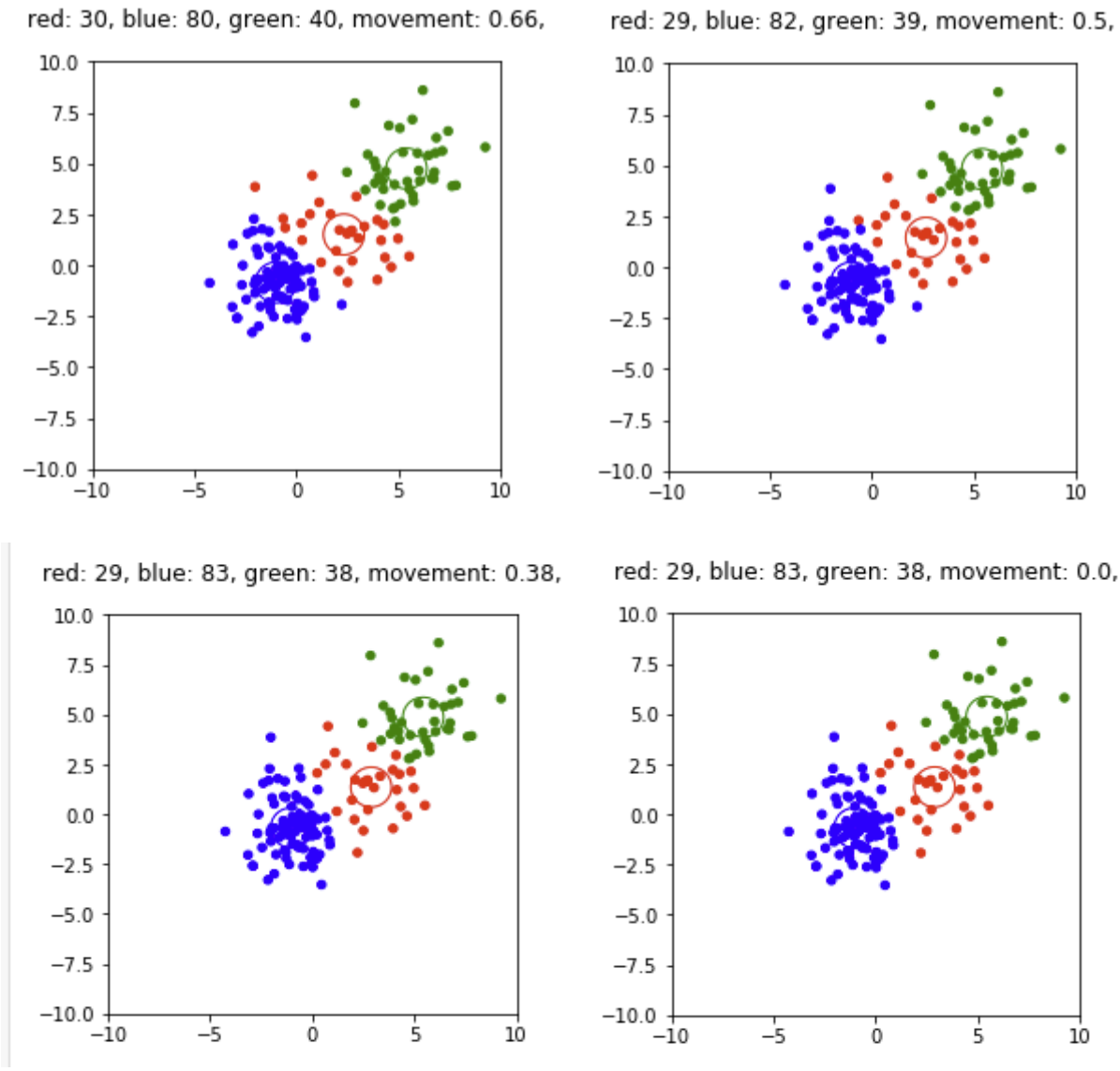
The movement has drastically decreased while the red and green clusters swap data points.



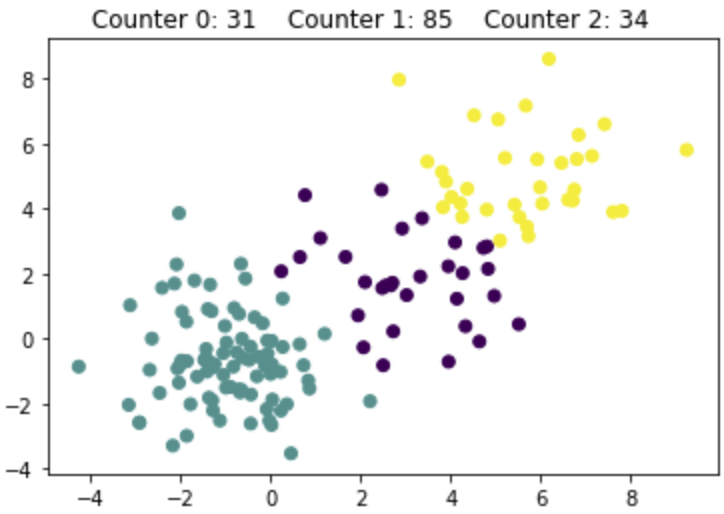
Iterations 5 through 10







Finally! In the tenth iteration, the relative movement has stopped. Compared with the results of the k-means application, the results are very similar. The application version distributes four data points differently. Four are moved from the one cluster and distributed evenly over the two other clusters.

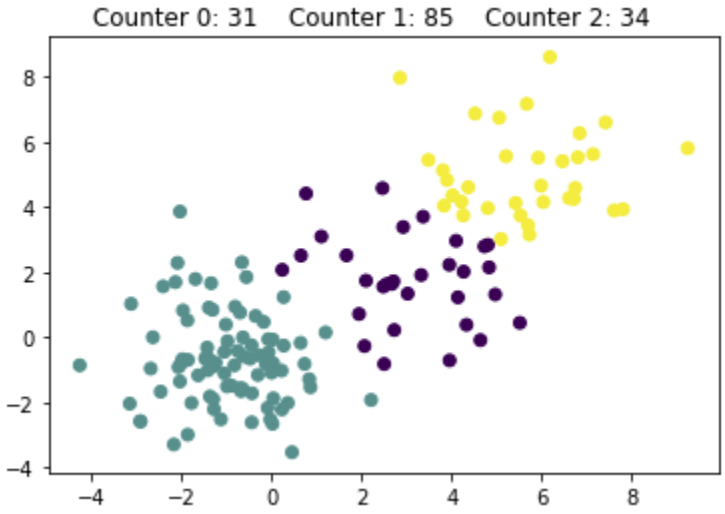


Part 2: Software Familiarization

k-means: KMeans from sklearn.cluster

The difference between the implementation is the tolerance of centroid movement. The implementation rounded the movement to two decimal places and stopped when the movement was less that 0.01. It suspected that the application was more precise with convergence.

Relative to the implementation, the code for the application was very brief:



```
# Compare to scikit learn.
def call_library():
    # Count members in each cluster.
    kmeans_counts = KMeans(n_clusters=3, random_state=0).fit(X)
    counters = [0] * 3
    for count in kmeans_counts.labels_:
        counters[count] += 1

    # Format title
    s = ' '
    for i in range(0, len(counters)):
        s += 'Counter ' + str(i) + ': ' + str(counters[i]) + '     '

    # Plot the predictions.
    kmeans_predict = KMeans(n_clusters=3, random_state=0).fit_predict(X)
    a = []
    b = []
    for x in X:
        a.append(x[0])
        b.append(x[1])
    plt.scatter(a, b, c=kmeans_predict)
    plt.title(s)
    plt.show()
```

Part 3: Applications

Kaushik Raghupathi authored web page "10 Interesting Use Cases for the K-Means Algorithm" located at https://dzone.com/articles/10-interesting-use-cases-for-the-k-means-algorithm. The goal of k-means is to find clusters in the data. Characteristic of the following examples, is that there are many homogeneous data points. Just about any grouping can be done. The drawback is that the number of groups is arbitrarily specified. A guess. Visualization of data points could indicate potential number of groups.

1. **Document Classification** Data points could be based on tags, topics, and even the frequency of words.
2. **Delivery Store Optimization** Locate delivery facilities near centroids of groups of regular customer.
3. **Identifying Crime Localities** Crime-prone areas can be identified. Data points could be location of crimes, types of crimes, and time-of-day.
4. **Customer Segmentation** Customer profiles could yield multiple classes of data. Clustering this data could discover new customer segment or market segment.
5. **Fantasy League Stat Analysis** Why just fantasy league? The book and movie "Money Ball" demonstrates how analytic tools like k-means can reveal yet unseen behaviors.
6. **Insurance Fraud Detection** Patterns of fraud could be detected by using historical data.
7. Rideshare Data Analysis Not unlike the "Delivery Store Optimization", instead of delivering good, transportation services are provided.
8. **Cyber-Profiling Criminals** Like the "Identifying Crime Localities", instead of finding locations, criminals in a network of associations can be discovered by clustering interactions.
9. **Call Record Detail Analysis** This is the cellular telephone company spying on us. The article says it is for customer demographics and determining services that would be desired. However, there have many news stories about telecommunication companies selling this analysis and data for an additional revenue stream.
10. **Automatic Clustering of IT** Alerts Log files generate massive amounts of messages. To see a pattern to system failure, a k-means application could cluster these messages. The system administrator could use these clusters to identify systemic problems.

Section G: Gaussian Mixture Model

Part 1: Implementation

Algorithm

The algorithm for this assignment is relatively simple:

- Step 1: Retrieve data points.
- Step 2: Create the arbitrary number of clusters at random locations.
- Step 3: Assign each data point to all clusters based on probabilities.
- Step 4: Compute the centroid for the cluster using the weighted data points.
- Step 5: Repeat "Step 3" and "Step 4" until the classification of data points stabilizes or the limit to the number of iterations is exceeded.

Implementation proved to be daunting. Results are not included due to lack of examples and guidance. One of the challenges was with a negative determinant. An assumption was made to use the absolute value of $|\Sigma_c|$ to be used.

Data Structures

The data points (X) and cluster (C) data structures used in k-means were also utilized in GMM. Additional structures are two-dimensional matrices. They are organized by cluster and then by data point. They are probability (P) and likelihood (L).

Code Optimization

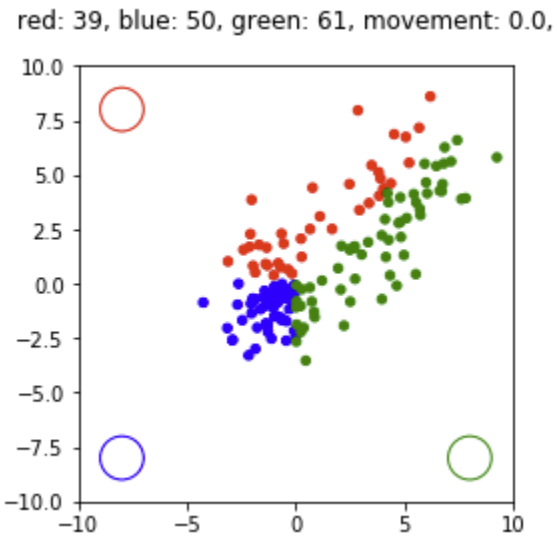
There was no optimization regarding the coding. However, there was optimization from a human perspective. Using a Jupyter Notebook, all functions were defined as functions. There was a single coding block that was the main routine that called the functions. The advantage of the Jupyter Notebook is debugging. A function in question could be position next to the main routine. This allowed for code modification and testing without having through many lines of code.

Even though the arguments of call functions are by reference, the coding philosophy of pass arguments and return a result set. This allows anyone doing maintenance to see the data flow.

Finally, a simple debug feature was added. The first parameter of all functions was a Boolean flag. When true, extensive printing of variables is done. False value suppresses all messages.

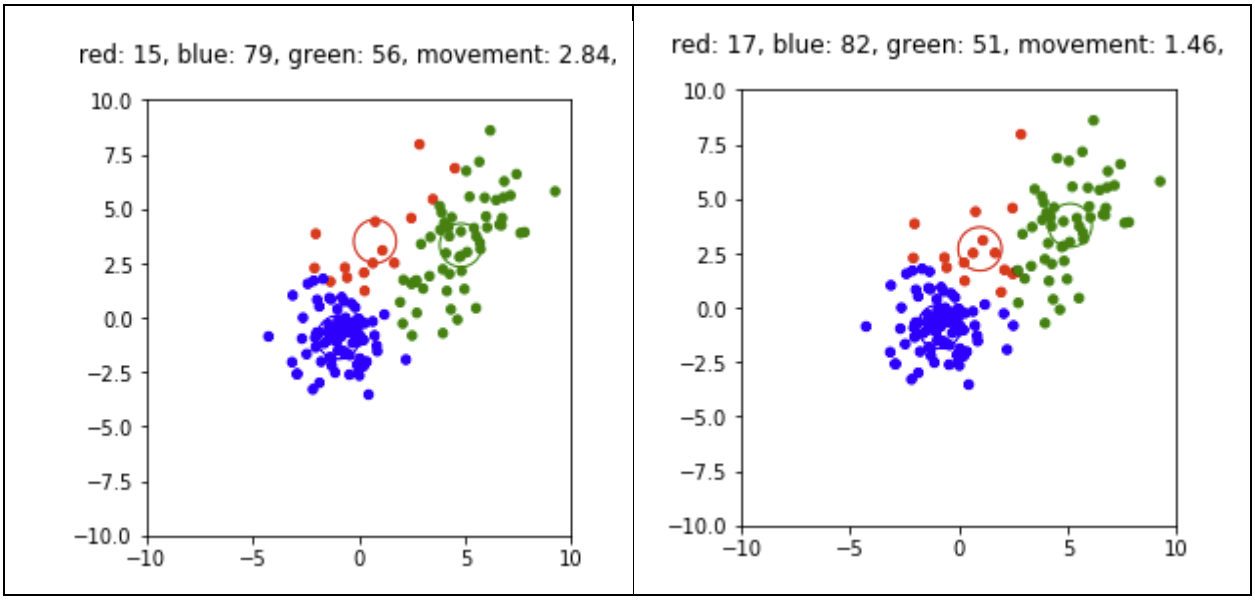
Iteration 1

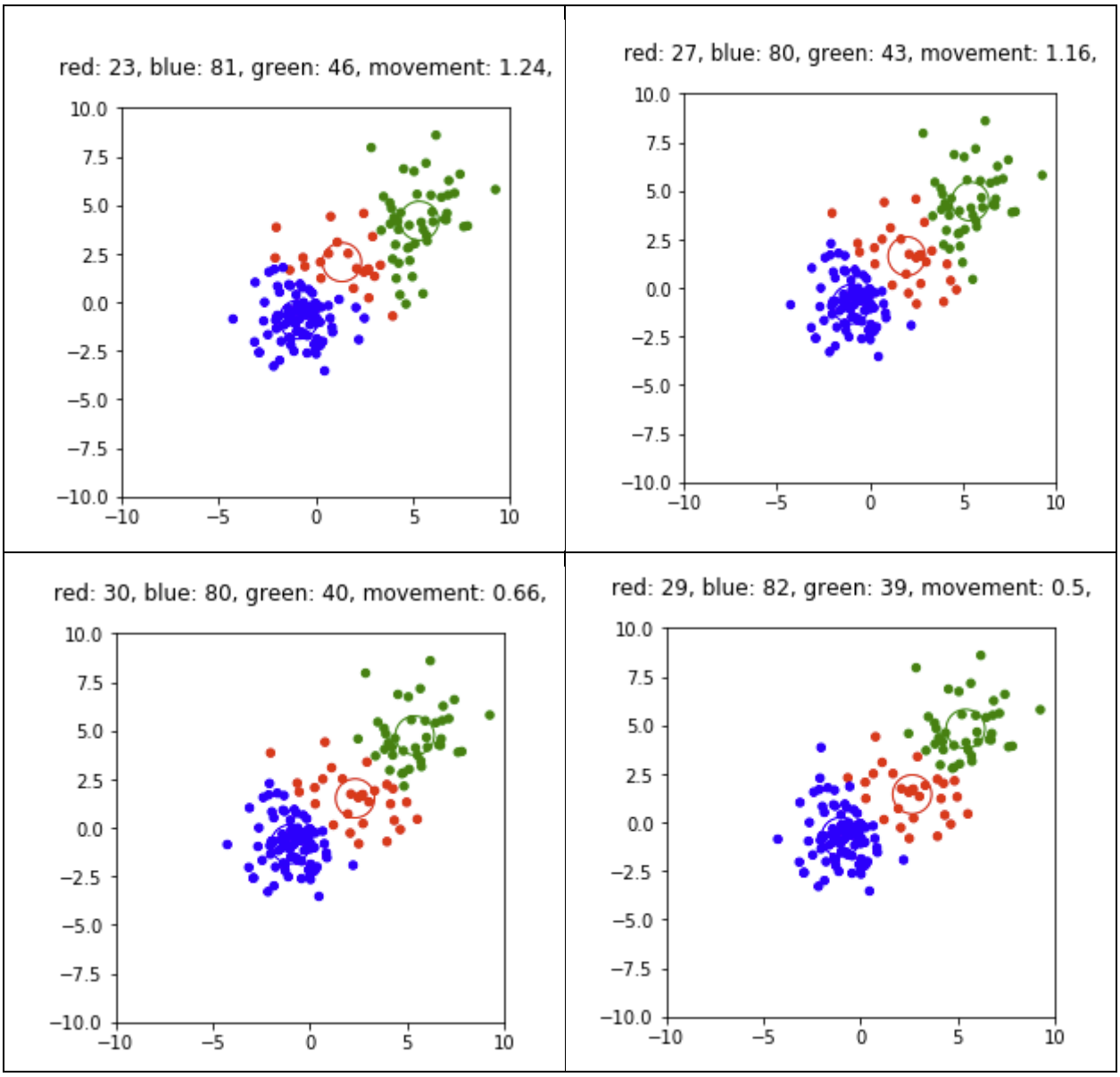
As done with k means, the initial centroids were position away from each other and away from the data points to emphasize changes in clustering.

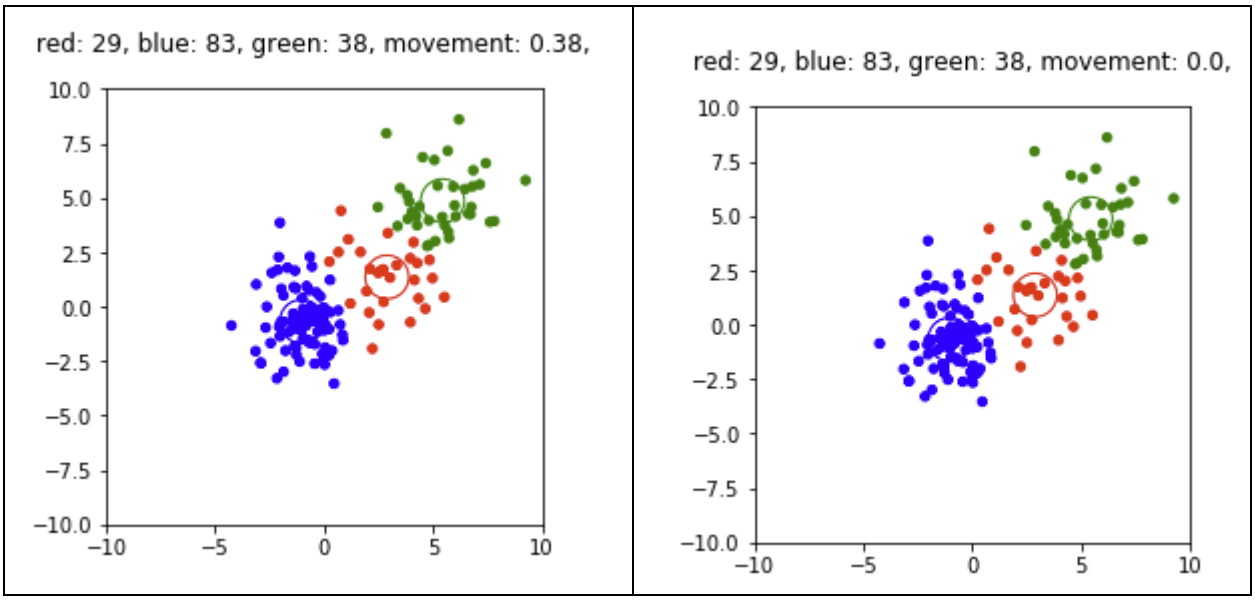


Iterations 2 through 9

As centroids move, the clusters swap data points with the other clusters. The process continues until the classification of data points stabilizes. This termination criteria is different from the k-means criteria of minimum centroid movement. The reason for the different criteria, well, what is life without whimsy?




red: 23, blue: 81, green: 46, movement: 1.24,
red: 27, blue: 80, green: 43, movement: 1.16,
red: 30, blue: 80, green: 40, movement: 0.66,
red: 29, blue: 82, green: 39, movement: 0.5,



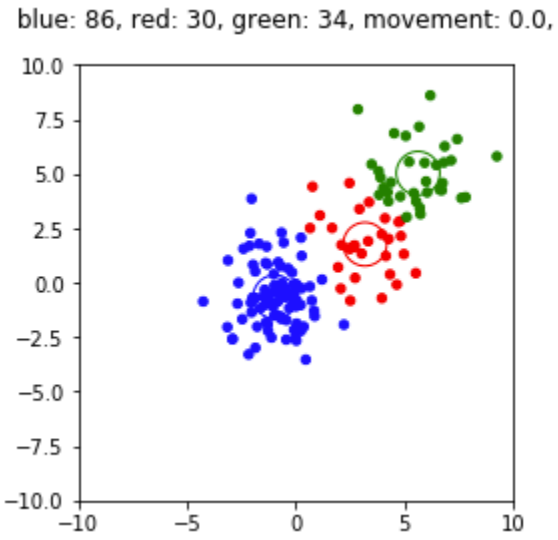
Even though the criteria for stopping was the stabilization of the classification counts, the final iteration had no further movement of centroids.

Part 2: Software Familiarization

GMM: Mixture from sklearn

Iteration 1

The code for using this library was relatively simple. The following table shows the tabulations of all methods.



data points	blue	red	green
k-means implementation	83	29	38
k-means sklearn k-means	85	31	34
GMM implementation	83	29	38
GMM sklearn mixture	86	30	34

None of the results are the same. However, they are very similar. The k-means implementation is the most different. This is likely caused by rounding and limiting the number of iterations.

GMM: Program Structure.

```
# Code copied from "Python Data Science Handbook" by Jake VanderPlas.
# Modified to accept custers.csv and cluster plotting
import itertools

import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
import matplotlib as mpl

from sklearn import mixture

# Number of samples per component
n_samples = 500

# Generate random sample, two components
np.random.seed(0)
C = np.array([[0., -0.1], [1.7, .4]])
X = np.r_[np.dot(np.random.randn(n_samples, 2), C),
          .7 * np.random.randn(n_samples, 2) + np.array([-6, 3])]

# Format data from cluster.csv to be a numpy ndarray.
X1 = load_X()
X2 = []
for x in X1:
    X2.append(x[0])
    X2.append(x[1])
shape = (150, 2)
buffer = np.array(X2)
X = np.ndarray(shape,buffer=buffer)

# Fit a Gaussian mixture with EM using three components.
gmm = mixture.GaussianMixture(n_components=3, covariance_type='full').fit(X)

# Create clusters for plotting.
colors = ['blue','red','green']

# Create control totals
T = {"blue":0, "red":0, "green":0, "movement":0.0}

# Create clusters container.
C = []
for color in colors:
    C.append({"color":color,"X":[]})

# Create predictions.
p = gmm.predict(X)

# For each prediction, update a cluster and control totals.
for i in range(0,len(p)):
    j = p[i]
    C[j]["X"].append(X[i])
    if j == 1:
        T["red"] += 1
    elif j == 2:
        T["green"] += 1
    else:
        T["blue"] += 1

# Calculate new centroids for the clusters.
C = calculate_centroids(C)

# Plot centroids clusters, and control totals.
plot_clusters(C,T)

```

Part 3: Applications

K-means is simple and simple to understand in a one or two-dimensional world. However, the world is complex with many interactive dimensions. Euclidian distances are not realistic. What is realistic is the frequent occurrence of normal distribution. Gaussian Mixture Model is useful be it assumes normal distributions of dimensions. It is better suited for a complex world.

Kshitij Maurya, Content Manager at Quantinsti, identifies an application that benefits from GMM-EM: **Asset return prediction**. Due to the vast number of variables, it is nearly impossible to manually design a model that makes predictions. GMM is multivariant, hence it could identify possible outcomes with the associated Gaussian distribution.

Covid-19 I suspect epidemiology has multivariate data sets for this pandemic.