[Project #3]

EM for GMM

[변지현, 20180298]

# Introduction

본 프로젝트는 EM 알고리즘을 활용하여 2차원 벡터들을 분류하고, 생성되는 가우시안 분포에 따라 벡터들이 어떻게 분류되는 과정을 시각화 하는 것을 목표로 한다.

EM 알고리즘은 관측되지 않는 잠재 변수에 의존하는 확률 모델에서 최대 우도(maximum likelihood)나 최대 사후확률(maximum a posterioir)을 갖는 반복적인 알고리즘이다. EM 알고리즘은 모수에 관한 추정값으로 로그우도(log likelihood)의 기대 값을 계산하는 E(Expectation)단계와 이 기대 값을 최대화하는 모수 추정값들을 구하는 M(Maximization) 단계를 번갈아 가면서 적용한다.

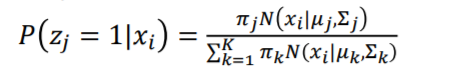
본 프로젝트에서는 1800개의 2차원 백터를 무작위로 분류하여 3개의 가우시안 분포를 생성한 뒤 이 결과를 더 이상 분류의 변화가 없을 때까지 EM알고리즘을 반복하여 최종 결과의 3차원 밀도 함수 및 벡터들의 포함관계에 대한 산점도를 확인한다. 또한 중간 결과들의 차원 밀도 함수와 산점도를 확인하여 가우시안 분포와 벡터들의 분류가 어떻게 변화하는지 확인한다.

# Algorithm

본 알고리즘은 크게 초기화, Expectation(이하 E단계), Maximization(이하 M)단계로 구분된다.

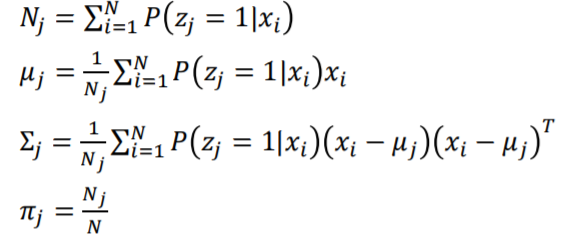
초기화 단계에서는 1800개의 벡터들을 무작위로 3개의 집합으로 분류를 한다. 그리고 각 집합들의 평균과 공분산 가중치를 계산한다. 이때 임의의 집합 의 가중치 는 모든 벡터의 개수를 , 집합 에 포함된 벡터의 개수를 라고 했을 때, 의 값을 갖는다.

E 단계에서는 각 집합에 대해 계산된 파라미터들(평균, 공분산, 가중치)들을 통해 각각의 가우시안 분포를 생성한다. 이 후, 모든 분포에 대해 각 벡터들의 최대 우도를 계산하여 가장 높은 우도를 갖는 분포의 집합으로 이동시킨다. 이 때 사용하는 최대 우도를 구하는 공식을 아래와 같다.



해당 과정을 모든 벡터들에 대해 반복하며 모든 벡터들을 가장 우도가 높은 집합으로 이동시킨다.

M 단계에서는 E 단계에서 새롭게 갱신된 각 집합들의 파라미터 정보를 갱신한다. 임의의 집합 의 평균을 , 공분산 행렬을 , 가중치를 라고 할 때, 이를 구하는 방법은 아래와 같다.



최종적으로는 초기화 과정 후 E단계, M 단계를 더 이상 모든 집합에 변화가 없을 때까지 반복하여 모든 접들이 가장 우도가 높은 집합에 속하도록 한다.

# Implementation Details

## Initialize

|  |
| --- |
| # initialize  gaussian\_set = [[], [], []]  expected\_in = [[0 for \_ in range(K)] for \_ in range(N)]  mean = []  cov = []  weight = []  for index, point in enumerate(points):  j = random.randint(0, 2)  gaussian\_set[j].append(point)  expected\_in[index][j] = 1  for index, gaussian in enumerate(gaussian\_set):  mean.append(np.mean(gaussian, axis=0))  cov.append(np.cov(np.array(gaussian).T))  weight.append(len(gaussian) / N) |

위 코드는 초기화 과정을 수행하는 소스 코드의 일부이다. K=3, N=1800으로 각각 가우시안의 개수, 모든 점들의 개수를 의미한다. 첫번째 반복문을 실행하며, 모든 점들에 대해 무작위로 3개의 집합에 배정한다. 이후 각 집합에 대해 평균, 공분산 행렬, 가중치를 계산한다.

expected\_in은 현재 벡터들이 어느 집합에 포함되어 있는지를 저장하는 2차원 배열로 expected\_in[0][0] == 1 일 경우, 인덱스가 0인 벡터가 0번 집합에 포함됨을 의미한다.

## Expectation

|  |
| --- |
| # Expectation          rvs = []          new\_expect = [[0 for \_ in range(K)] for \_ in range(N)]          for index in range(K):              rvs.append(multivariate\_normal(mean[index], cov[index]))          gaussian\_set = [[], [], []]          for i in range(N):              whole\_probability = 0              probabilities = []              for k in range(K):                  probabilities.append(weight[k] \* rvs[k].pdf(points[i]))              whole\_probability = sum(probabilities)              highest = -1              gaussian\_index = 0              for j in range(K):                  probability = probabilities[j]/whole\_probability                  if highest < probability:                      highest = probability                      gaussian\_index = j              gaussian\_set[gaussian\_index].append(points[i])              new\_expect[i][gaussian\_index] = 1 |

위 코드는 E 단계를 구현한 소스코드의 일부이다. 각각의 집합에 대해 계산한 평균과 공분산 행렬을 scipy 패키지의 multivariate\_noraml 메소드를 사용하여 각 집합의 확률 밀도 함수를 구한다. 이 후 각 집합의 확률 밀도 함수을 통해 모든 벡터들의 좌표의 우도를 계산한다. 그 중 가장 높은 우도를 갖는 집합에 해당 벡터를 집합에 재 배정한다.

new\_expect는 새로 배정된 벡터들이 어느 집합에 포함되어 있는가를 저장하는 2차원 배열로 expected\_in 배열과 비교하여 알고리즘의 종료 시점을 결정하는데 사용된다. expected\_in과 비교하여 nex\_expect가 차이가 없을 경우 알고리즘을 종료한다.

## Maximization

|  |
| --- |
| # Maximization          mean = []          cov = []          weight = []          for index, gaussian in enumerate(gaussian\_set):              mean.append(np.mean(gaussian, axis=0))              cov.append(np.cov(np.array(gaussian).T, bias=True))              weight.append(len(gaussian) / N) |

위 코드는 M 단계를 구현한 소스 코드의 일부이다. E 단계에서 새로 생성된 집합들에 대해 평균, 공분산 그리고 가중치를 갱신하는 과정이 포함된다.

# Experimental Results

본 프로젝트는 EM 알고리즘이 반복되는 과정에서 5번의 반복마다 3차원 확률 밀도 함수와, 점들의 산점도를 확인하도록 하였다.

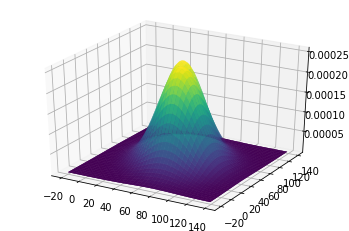
본 실험에서는 총 40회의 반복 학습이 일어났음을 확인하였고 이에 따라 초기화 직후, 그리고 각 수행의 5번 반복마다 평균과 공분산행렬, 가중치, 해당 시점의 확률분포, 벡터들의 산점도를 확인하여 학습이 수행됨에 따른 변화를 확인하였다.

## 초기화 직후

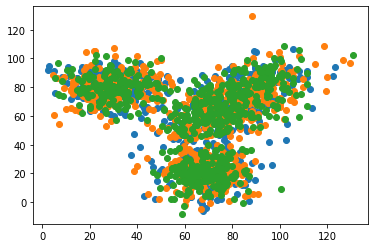
파라미터(순서대로 0, 1, 2 번 분포의 파라미터이다)

|  |  |  |  |
| --- | --- | --- | --- |
| index | 평균 | 공분산 행렬 | 가중치 |
| 0 | [61.80204217, 59.3058191 ] | [[ 610.19798283, -145.51365481],  [-145.51365481, 755.89543356]] | 0.31666666666666665 |
| 1 | [63.83118395, 58.82180685] | [[ 623.27665468, -106.29465289],  [-106.29465289, 734.30210255]] | 0.3327777777777778 |
| 2 | [63.74562615, 58.72652591] | [[ 552.04786591, -107.22993045],  [-107.22993045, 726.1651768 ]] | 0.35055555555555556 |

위의 파라미터를 기반으로 시각화한 확률 밀도함수의 이미지는 아래와 같다. 평균값과 가중치의 차이가 적어 하나의 분포인 것과 유사한 이미지가 출력됨을 확인할 수 있다.



또한, 무작위로 초기화한 벡터들을 집합에 포함관계에 따라 다른 색으로 표시한 것이다. 모든 점들이 균일하게 분포한 것을 확인할 수 있다.

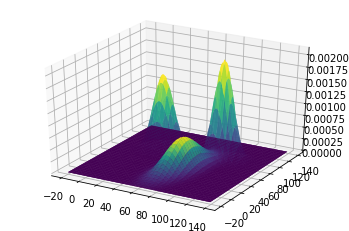


## 5회 반복

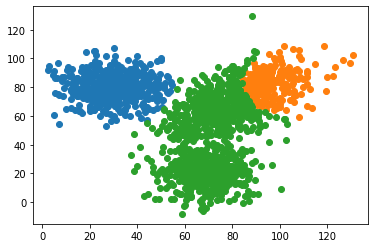
파라미터

|  |  |  |  |
| --- | --- | --- | --- |
| index | 평균 | 공분산 행렬 | 가중치 |
| 0 | [29.85444342, 80.0724885] | [[1.13104048e+02, 5.08769394e-02],  [5.08769394e-02, 8.63908854e+01 | 0.27666666666666667 |
| 1 | [96.09479926, 81.98184141] | [[72.64796992, 21.89401498],  [21.89401498, 85.25050774]] | 0.1388888888888889 |
| 2 | [71.09726301, 43.46336256 | [[72.64796992, 21.89401498],  [21.89401498, 85.25050774]] | 0.5844444444444444 |

아래는 5회 반복 후 확률 밀도 함수를 시각화한 이미지이다. 초기화 직후와 달리 3개의 분포를 시각적으로 확인할 수 있다.



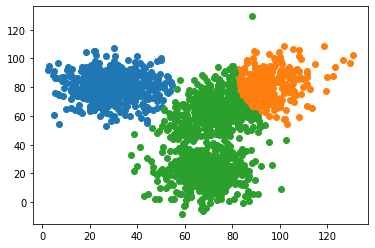
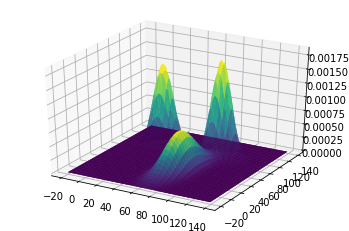
벡터들의 산점도 또한 3개의 색상이 3개의 구역으로 뭉쳐있음을 확인할 수 있다.



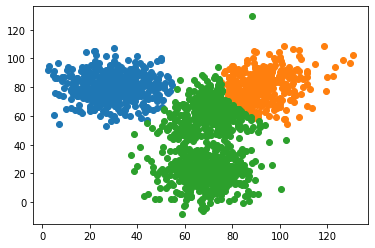
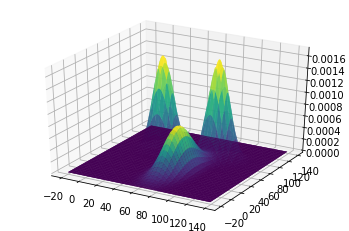
## 10 ~ 35회 반복

10 ~ 35회를 반복함에 따라 가우시안 분포와 산점도가 변화하는 과정을 시각화하는 이미지는 아래와 같다. 반복횟수가 증가함에 따라 초록색 군집과 주황색 군집이 분리되는 과정을 확인할 수 있다.

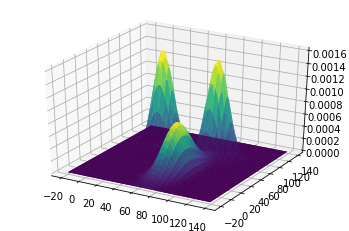
**10회 반복 시 확률 밀도 함수 및 산점도**

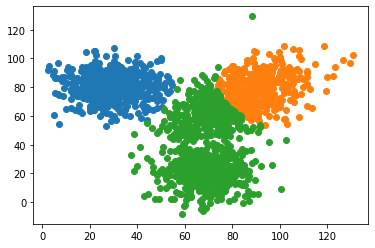


**15회 반복 시 확률 밀도 함수 및 산점도**

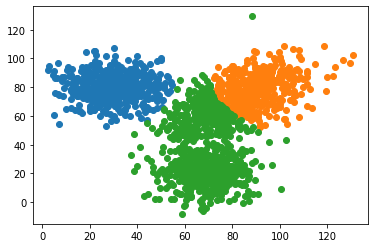
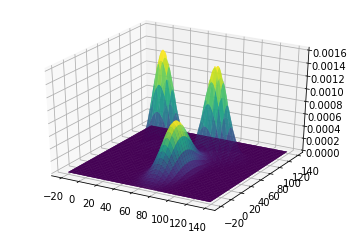


**20회 반복 시 확률 밀도 함수 및 산점도**

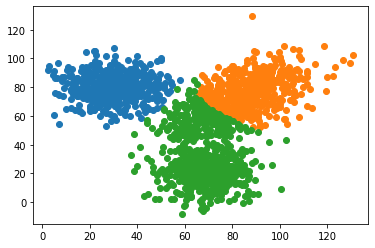
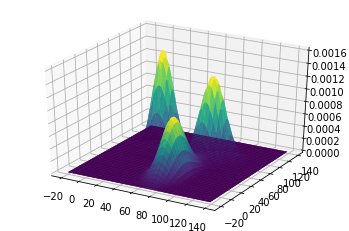




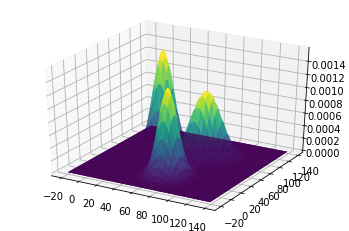
**25회 반복 시 확률 밀도 함수 및 산점도**



**30회 반복 시 확률 밀도 함수 및 산점도**



**35회 반복 시 확률 밀도 함수 및 산점도**

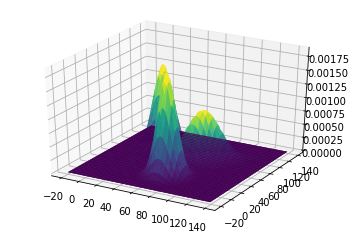


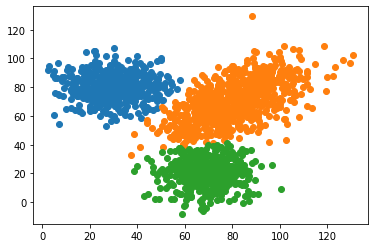
## 최종(40회 반복)

파라미터

|  |  |  |  |
| --- | --- | --- | --- |
| index | 평균 | 공분산 행렬 | 가중치 |
| 0 | [29.89451311, 80.15232825] | [[114.86710282, 1.45824912],  [ 1.45824912, 84.99958076]] | 0.27666666666666667 |
| 1 | [80.14406744, 70.07729836] | [[210.0180075 , 121.98265308],  [121.98265308, 195.54442105]]85.25050774]] | 0.4461111111111111 |
| 2 | [69.0228049 , 19.85388174] | [[87.24272942, 1.72141683],  [ 1.72141683, 83.05191787]] | 0.2772222222222222 |

아래의 이미지는 위의 파라미터 값을 기반으로 최종적으로 수행된 모델의 가우시안 분포를 시각화 한 이미지와 산점도를 출력한 것이다. 위 과정들과 비교하여 3개의 가우시안 분포가 뚜렷하게 시각화 된 것을 볼 수 있으며, 산점도 또한 3개의 집합이 색깔이 거의 겹치지 않게 분류된 것을 확인할 수 있다.





# Conclusion

본 프로젝트에서는 EM 알고리즘을 사용하여 2차원 벡터들을 군집화하고 그 과정을 시각화하여 분석하였다. 초기화 과정에서의 평균은 각각 [61.80204217, 59.3058191], [63.83118395, 58.82180685], [63.74562615, 58.72652591]으로 3개의 군집 모두 비슷한 평균을 가졌고, 가중치 또한 각각 [0.31666666666666665, 0.3327777777777778, 0.35055555555555556]으로 비슷한 값을 가져 결과적으로 3개의 가우시안이 큰 차이를 보이지 않는 것을 확인할 수 있었다.

하지만, 학습이 진행됨에 따라 3개의 가우시안 분포가 점차적으로 뚜렷해짐을 확인할 수 있었는데, 5회의 학습을 진행한 시점의 경우 평균이 각각 [29.85444342, 80.0724885 ] [96.09479926, 81.98184141], [71.09726301, 43.46336256]으로 초기화 직후에 비해 각각의 분포의 평균의 차이가 커진 것을 확인할 수 있었고, 벡터들의 군집화 또한 개선된 모습을 확인할 수 있었다.

최종 모델의 경우 평균이 각각 [29.89451311, 80.15232825], [80.14406744, 70.07729836], [69.0228049 , 19.85388174]으로, 산점도를 확인 해본 결과 거의 모든 벡터들이 다른 군집과 겹침이 없이 분류가 된 것을 확인할 수 있었다.

본 시험에서 진행하였을 때 알고리즘이 종료되기 전까지 총 40번의 학습이 진행되었는데, EM 알고리즘은 초기 군집의 파라미터에 영향을 받기 때문에 무작위로 초기화를 진행을 한 만큼 그에 대한 영향을 받은 것으로 보인다. 따라서 초기 군집의 초기화를 수정하면 더 좋은 성능을 보일 것으로 기대되는데, 무작위로 초기화하는 방법 대신, greedy 알고리즘을 도입하여 가까운 점들 끼리 집합을 생성하여 초기 군집을 생성하는 방법 등을 사용함으로써 모델의 성능을 향상시킬 수 있을 것으로 보인다.