Density Estimation and Applications in Clustering and Segmentation

Michael Ferko
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, USA
Email: Mike.W.Ferko@gmail.com

***Abstract*—Density Estimation can be a parametric or non-parametric (or no parameter) a posteriori (posterior) probability distribution estimation technique. In this article we will explore the use of the Parzen Window, K-NN (Kernel Nearest Neighbor), MLE (Maximum Likelihood Estimation) and the LCG E-M Algorithm (Linear Combination of Gaussian Expectation/Estimation-step and Maximization-step algorithm) for density estimation to achieve data clustering and image segmentation via Bayesian Classification.**

Keywords—Bayesian Classifier, Pattern Recognition, Machine Intelligence, Probability & Statistics, Engineering, Parzen Window, K-NN, Maximum Likelihood Estimation, LCG E-M Algorithm, Density Estimation

I. Introduction

This project is split into two parts: Part 1 deals with parametric and non-parametric Density Estimation the EM Algorithm. Part 2 explores Density Estimation techniques a bit further with Bayesian classification.

II. PART 1: Density Estimations & EM

The Parzen-window method (also known as Parzen-Rosenblatt window method) is a widely used non-parametric approach to estimate a probability density function p(x) for a specific point p(x) from a sample p(xn) that does not require any knowledge or assumption about the underlying distribution. It is a nonparametric technique which can be used to fit each point to a probability distribution. When the Parzen window is scanned across a data set the weighted sum of each point contribution (center point estimation) becomes the normalized probability contribution at that data point. In fact as the window bandwidth h_n approaches zero the window kernel becomes the dirac delta function at the given data center point.

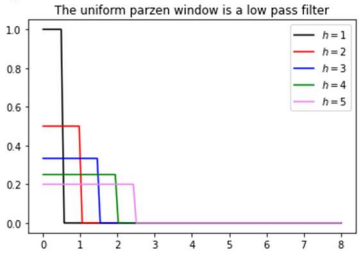


Figure 1: Parzen Window Approaches Dirac Delta Function as h_n approaches zero

For a hypercube of unit length 1 centered at the coordinate system's origin. What this function basically does is assigning a value 1 to a sample point if it lies within 1/2 of the edges of the hypercube, and 0 if lies outside (note that the evaluation is done for all dimensions of the sample point).

$$\phi(\boldsymbol{u}) = \begin{bmatrix} 1 & |u_j| \le 1/2\,; \quad j = 1, \ldots, d \\ 0 & \text{otherwise} \end{bmatrix}$$

If we extend on this concept, we can define a more general equation that applies to hypercubes of any length hn that are centered at x:

$$k_n = \sum_{i=1}^{n} \phi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)$$

where

$$\boldsymbol{u} = \left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)$$

- probability density estimation with hypercube kernel

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h^d}\phi\left[\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right]$$

where

$$h^d = V_n \quad \text{and} \quad \phi\left[\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right] = k$$

-probability density estimation with Gaussian kernel

$$p_n(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h^d}\phi\left[\frac{1}{(\sqrt{2\pi})^d h_n^d}\exp\left[-\frac{1}{2}\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n}\right)^2\right]\right]$$

Figure 2: The Unit Hypercube and The Parzen Window

Now that we understand the concepts of the parzen window let us implement them:

II.a. PART 1.1: Parzen Window and K-NN

Fit a nonparametric density estimate to the samples in "data.txt" using Parzen windows approach. You may use the following windows:

a. $\phi(x) = \begin{cases} 1/h & |x - x_i| \le h/2 \\ 0 & otherwise \end{cases}$

b. $\phi(x) = \begin{cases} \frac{h+x-x_i}{h^2} & x - h \le x \le x_i \\ \frac{h+x-x_i}{h^2} & x_i \le x \le x_i + h \\ 0 & otherwise \end{cases}$

c. $\phi(x) = \frac{1}{\sqrt{2\pi}h} e^{\frac{(x-x_i)^2}{2h^2}}$

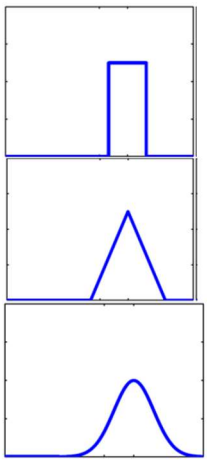


Figure 3: Uniform, Triangular and Gaussian Parzen Windows

When given a set of data the best way available to python 3 is to reformat a data file as a coma separated variable (.csv) file in Microsoft Excel. If the data does not have commas as would be needed for csv files we can simply import any text file with available spaces between data points and the auto fill feature of excel will create the csv file as needed. The data given in this project was received as a raw text file with spaces and was easily converted to a csv file, transposed and data framed. This data preprocessing allows for fluent and easy access to data manipulation. The data was then scatter plotted and a density histogram was imposed to better understand the data.

	Row	Values
0	0	0.531578
1	1	2.455880
2	2	0.182226
3	3	1.738233
4	4	1.662072
...	...	...
995	995	1.623437
996	996	0.397145
997	997	1.272125
998	998	0.084573
999	999	3.302696

1000 rows × 2 columns

Figure 4: Pandas Data Frame of data.csv

	Row	Values
count	1000.000000	1000.000000
mean	499.500000	1.019726
std	288.819436	0.996139
min	0.000000	0.010559
25%	249.750000	0.298978
50%	499.500000	0.703355
75%	749.250000	1.465840
max	999.000000	8.469454

Figure 5: Pandas Data Frame Description of data.csv

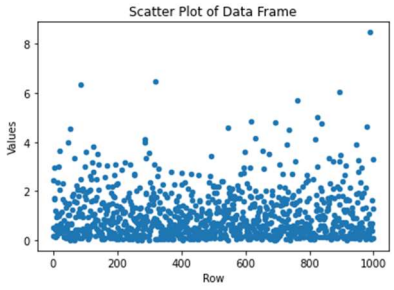


Figure 6: Scatter Plot of Data.csv

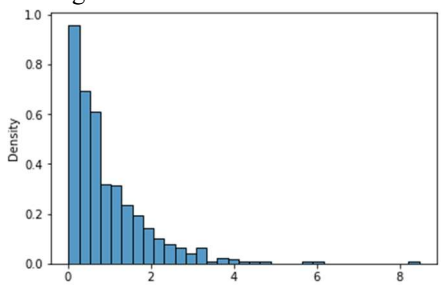


Figure 7: Histogram of data.csv

II.a.1. PART 1.1 i.: Parzen Window

i. Using different values of h = 0.01, 0.1 and 1.0, plot the nonparametric density estimates overlaid on the histogram of the data.

a. $$\phi(x) = \begin{cases} 1/h & |x - x_i| \le h/2 \\ 0 & otherwise \end{cases}$$

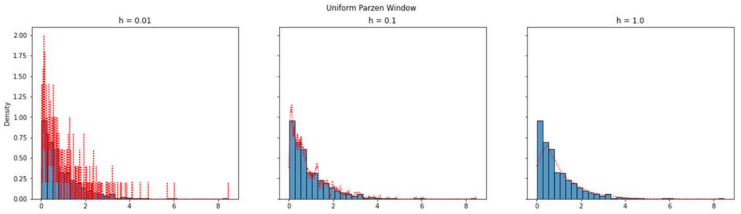


Figure 8: Uniform Parzen Window Density Estimates, from left to right: h = 0.01, 0.1, 1.0

From Figure 8 we can see the low bandwidth does not estimate the density very well.

b. $$\phi(x) = \begin{cases} \frac{h+x-x_i}{h^2} & x - h \le x \le x_i \\ \frac{h+x-x_i}{h^2} & x_i \le x \le x_i + h \\ 0 & otherwise \end{cases}$$

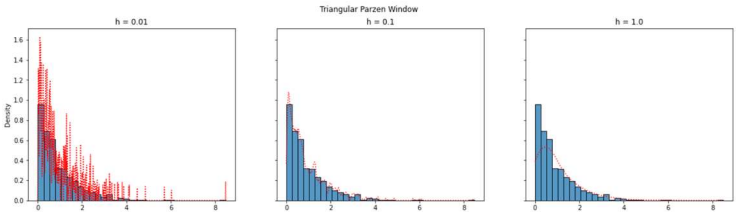


Figure 9: Triangular Parzen Window Density Estimates, from left to right: h = 0.01, 0.1, 1.0

c. $$\phi(x) = \frac{1}{\sqrt{2\pi}h} e^{-\frac{(x-x_i)^2}{2h^2}}$$

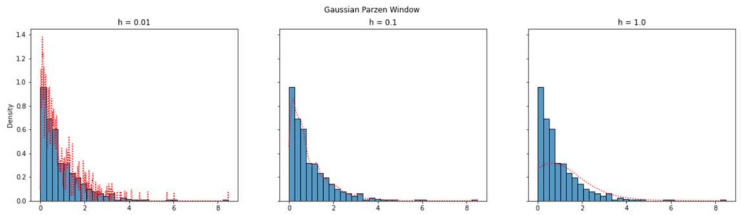


Figure 10: Gaussian Parzen Window Density Estimates, from left to right: h = 0.01, 0.1, 1.0

From the nine estimations above it is clear the medium bandwidth of 0.1 made the best density estimations and the gaussian distribution estimated the density better than the other two window types.

II.a.2. PART 1.2: K-NN

ii. Using window in (a) find k-NN nonparametric density estimates using different values of k = 1,10 and 30, plot the nonparametric density estimates overlaid on the histogram of the data.

Before looking at the nonparametric density estimation for the K-Nearest Neighbor density estimates; it is also important to know the K-NN algorithm can use various types of distance measures between points. The distance measure could be well known measures such as the Euclidean, Manhattan and Minkowski.

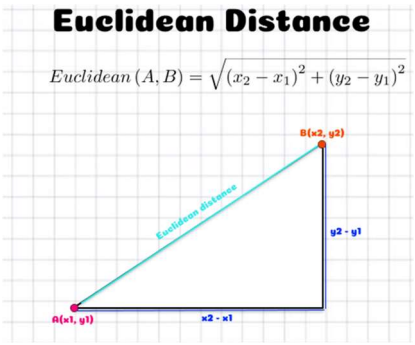


Figure 11: Euclidean Distance

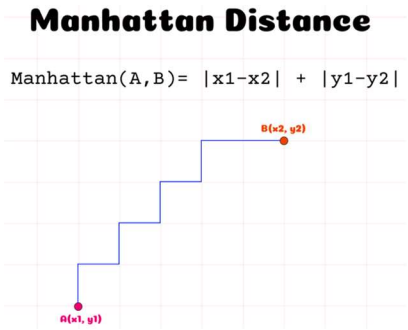


Figure 11: Manhattan Distance

$$Minkowski\,(A,B) = (\sum_{i=1}^{n} |(fa_i - fb_i)^p|)^{\frac{1}{p}}$$

Figure 12: Minkowski Distance

To implement the K-NN algorithm we think of each data point having the closest point in value to be the nearest neigbor and we find the distance measure in the blocked-off neighborhood of each selected data point. In this implementation we will be working with the Manhattan distance. Now when working with 1 dimensional data sets the Euclidean and Manhattan distance measures become the same since the square root of a square is the absolute value. When the distances are sorted we select the number of closest neighbors to a data point and this becomes a kernel in a similar way the parzen window was the kernel in the previous exercise. The difference here is that in the parzen window approach we used a set unit volume of data points. Where as in the K-Nearest Neighbor approach we use a set Kernel for the number of neighbors in a data point neighborhood. Where k in the following figure represents the number of neighbors.

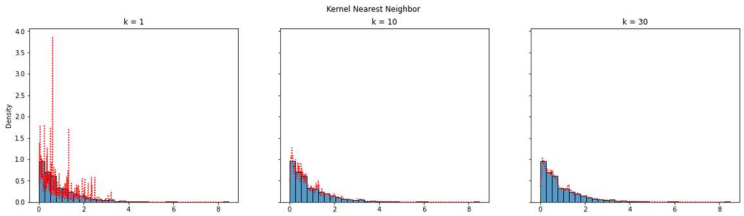


Figure 13: K-NN Density Estimates from left to right: k = 1, 10, 30

There seems to be an issue with the number of neighbors set to 1, but the rest of the estimations are fairly good.

II.b. PART 1.2: Maximum Likelihood Estimation (MLE)

The exponential density has a single parameter θ, where $p(\theta) = \theta e^{-\theta x}, \theta > 0\ and\ x > 0$. Given n samples $x_1, x_2, \dots, x_n$ that are drawn independently and identically from the exponential distribution.

II.b.1. PART 1.2 i.: Derive MLE for θ

i) Derive the maximum likelihood estimate for θ:

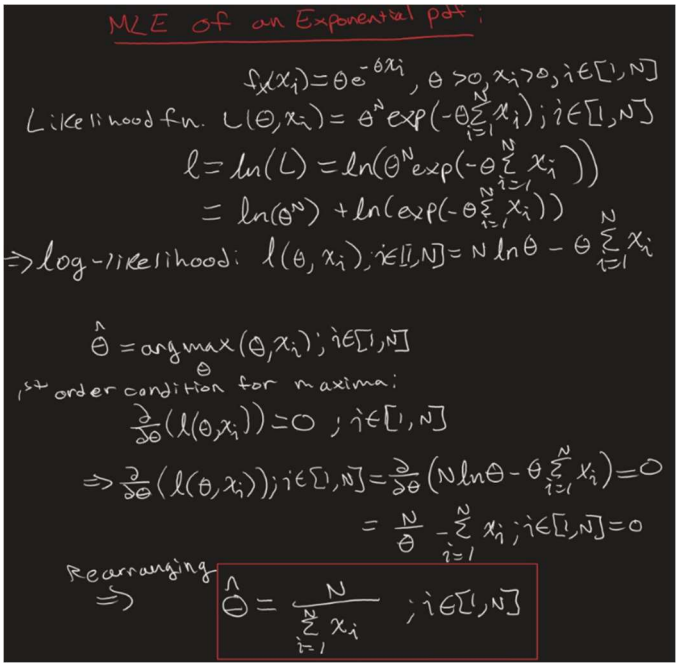


Figure 14: Parameter θ Derivation

II.b.2. PART 1.2 ii.: Compute MLE for θ

ii) Compute the maximum likelihood estimate for θ for the samples in "data.txt". Plot the maximum likelihood exponential density overlaid on the histogram of the data.

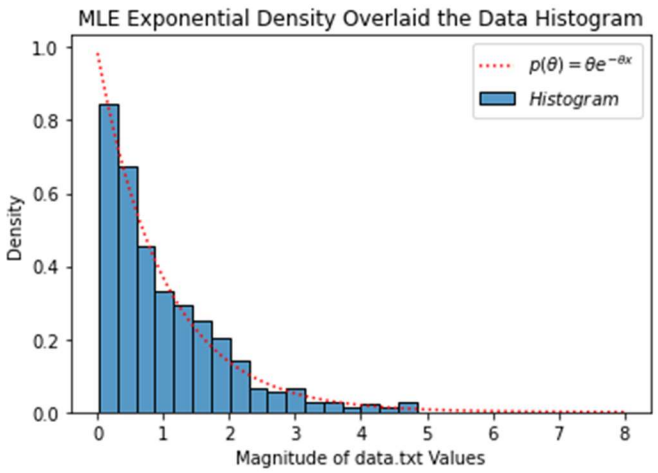


Figure 15: Parametric MLE Density Estimation of Data.csv

The Density estimation using the MLE parameter θ was far more accurate.

II.a.3. PART 1.3: K-Component Gaussian Mixture Model (GMM)

Consider the K-component Gaussian mixture model with parameters (K=3):

$$\alpha_1 = 0.5, \quad \mu_1 = 43, \quad \sigma_1 = 10$$
$$\alpha_2 = 0.2, \quad \mu_2 = 128, \quad \sigma_2 = 10$$
$$\alpha_3 = 0.3, \quad \mu_3 = 170, \quad \sigma_3 = 10$$

a. Write a function that takes as input a value for n and returns a 1×n vector x representing a random sample drawn from this mixture model.
b. Write a function "EM" that takes x and an integer K and returns the MLE of a Gaussian mixture model with K components, as computed by the EM algorithm.
c. In the code for the EM algorithm, include commands that show the K different Gaussians components as they are iteratively updated. (Similar to what has been done in the class.
d. Generate a sample from the Gaussian mixture model of size n = 1000. Call EM function with K = 3. Report the following:
    i. The estimates of the model parameters and how they compare to the true values
    ii. The effect of initial values and the number of iterations required until convergence
    iii. Plot the log-likelihood as a function of iteration number and verify that it is nondecreasing.

From Bayesian Classification on Parametric Class Probability Distributions [3]. We implement the uniform distribution generation inverse transform the standard uniform distribution to the standard normal distribution and affine transform to a desired mean and desired standard deviation. We do this for a given number of points for each mean and variance desired i.e. since K = 3 we generate the 3 transformed gaussians of a large number of points say 10,000. The data is then plotted for the scatter and histogram.

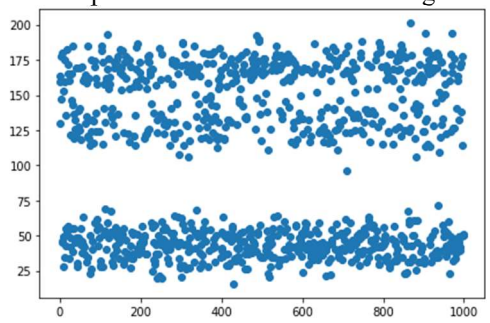


Figure 16: Scatter Plot of GMM

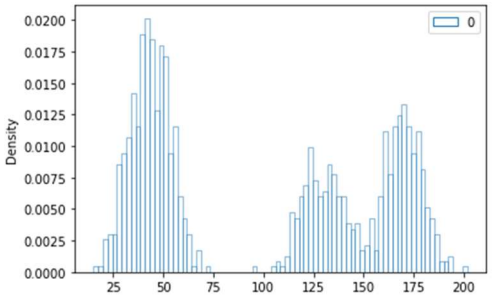


Figure 17: Histogram Plot of GMM

One parameter left out when the EM algorithm is described is how to define when the iterations of the EM should stop. After implementing a convergence criteria we can call an error which in my implementation was defined as "convergence_discriminant_toll," but really it is just the absolute difference between the previous and current log likelihood of an iteration where the initialization criteria requires negative infinity for python this is -sys.maxsize. The following are the results of the EM algorithm on the above GMM generated data. The error was set to 0.001.

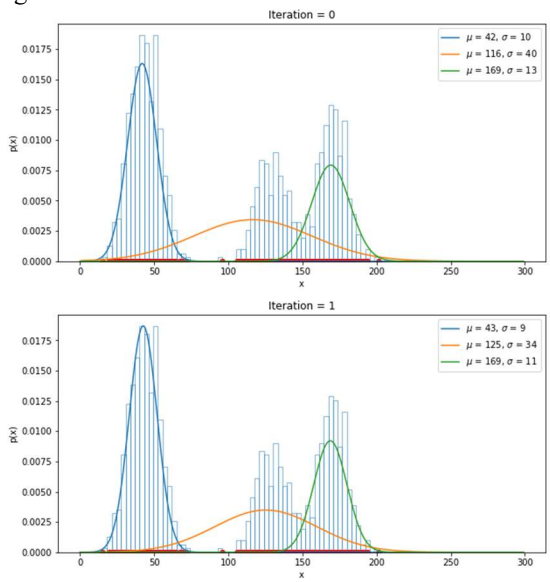


Figure 18: LCG EM on GMM: Iteration 0, 1

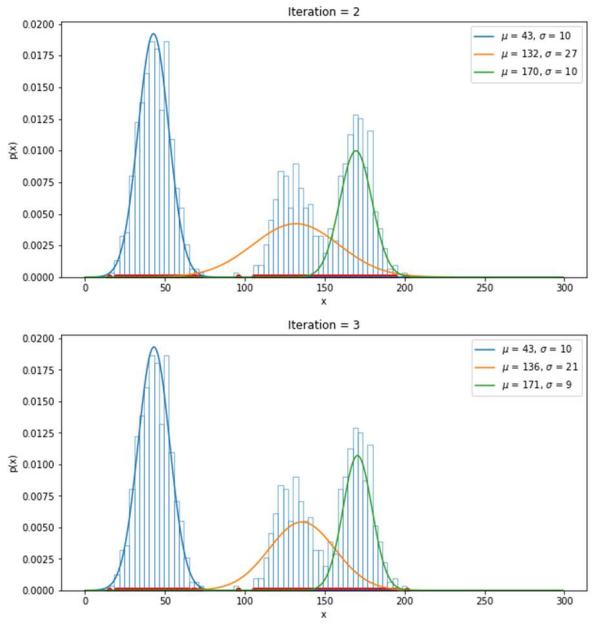


Figure 19: LCG EM on GMM: Iteration 2, 3

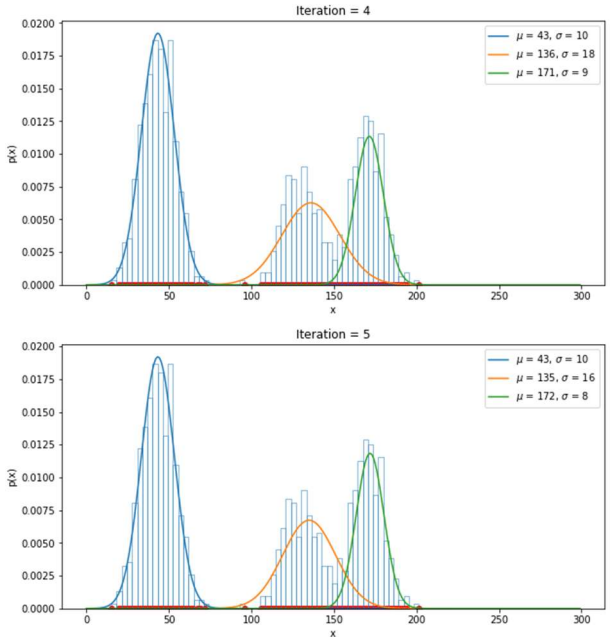


Figure 20: LCG EM on GMM: Iteration 4, 5

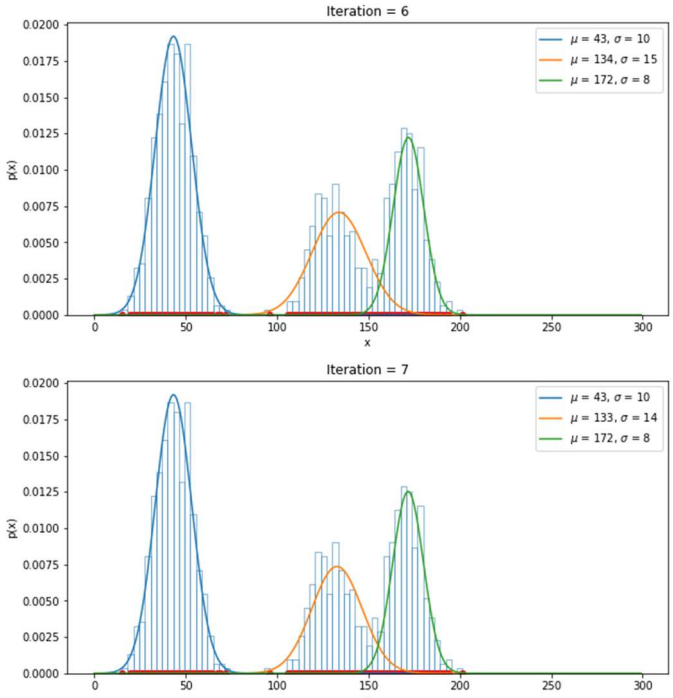


Figure 21: LCG EM on GMM: Iteration 6, 7

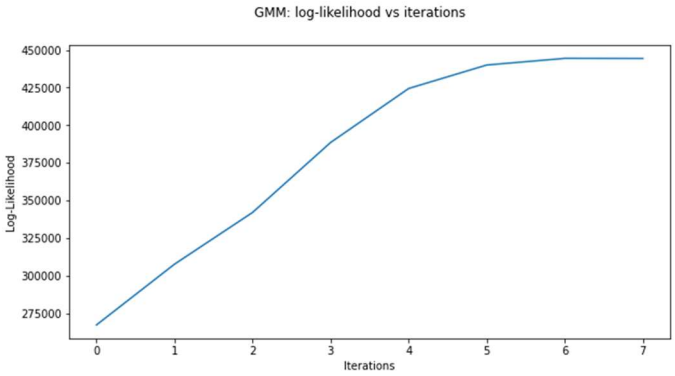


Figure 22: Log-likelihood vs. Iterations for GMM

From Figure 21 we can see the final iteration for the density estimation of the GMM using the EM algorithm is highly accurate!

II.a.4. PART 1.4: K-Component EM on Lung and Kidney

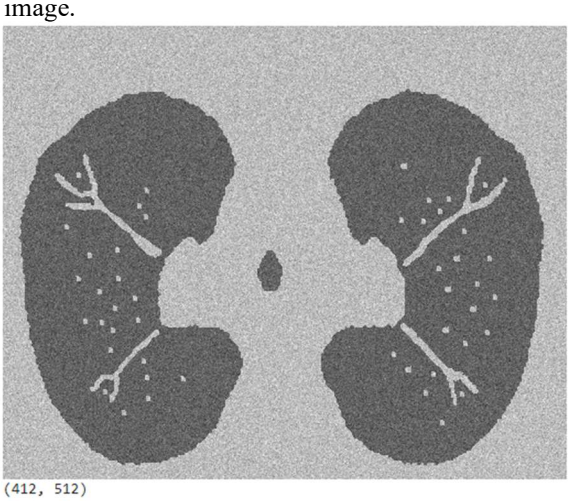
Using the EM algorithm, estimate the marginal densities of the classes in the following images:

a) "Lung.pgm"
b) "Kidney.pgm"

Get an initial cluster based on these estimations.

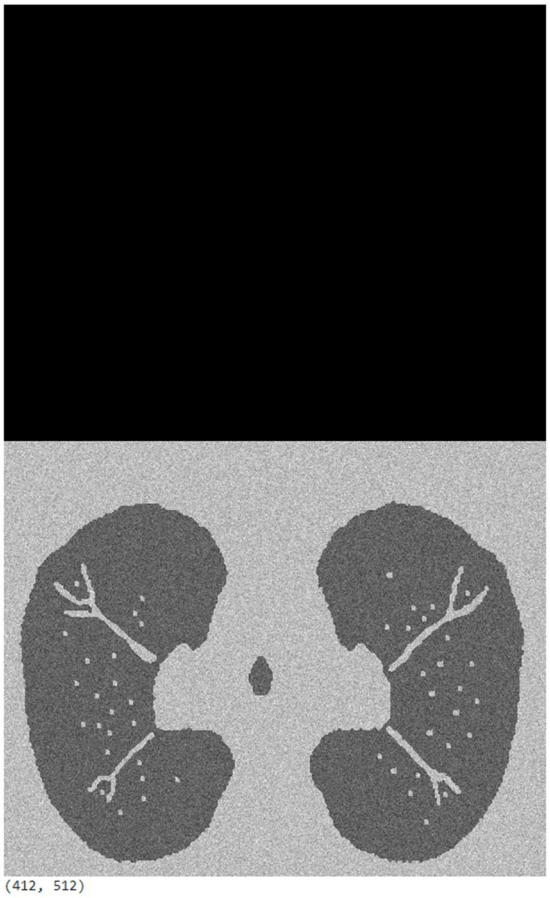
The images were imported using the opencv module and normalized by setting the base intensity to zero with 256 levels and a max intensity of 255 (i.e., $L = 2^n - 1 = 2^8 - 1 = 256 - 1 = 255$). The two image were converted to gray scale since the opencv module automatically interprets all

images with a red, green and blue channel depth. The following are the results of the lung image, normalization, reshaping, intensity plot and LCG EM algorithm ran on the lung image.



(412, 512)

Figure 23: Imported Lung Image



(412, 512)

Figure 24: Normalized Lung Image

```
[[211.]
 [156.]
 [194.]
 ...
 [175.]
 [194.]
 [187.]]
(210944, 1)
```

Figure 25: Lung Image Vector Reshaping for application

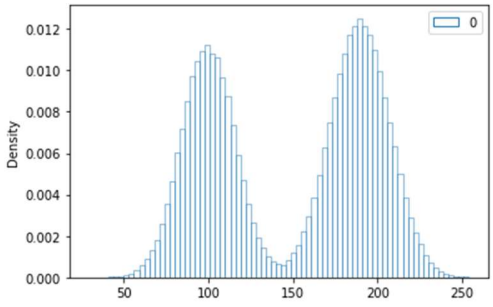
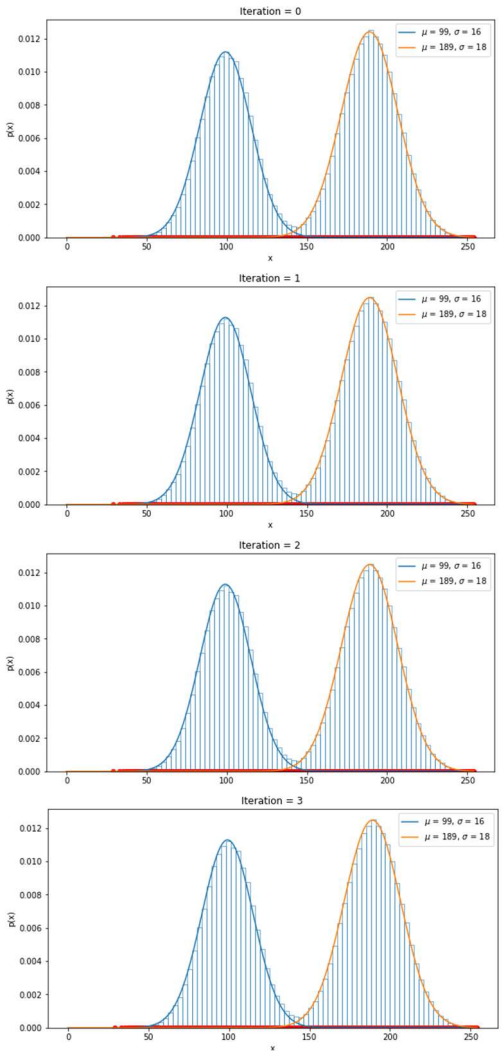


Figure 26: Lung Image Vector Histogram



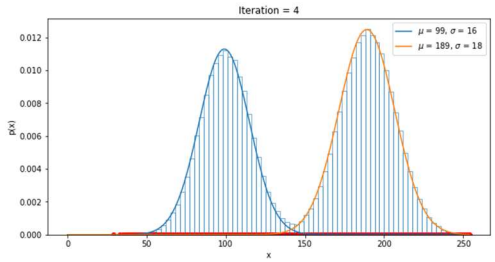
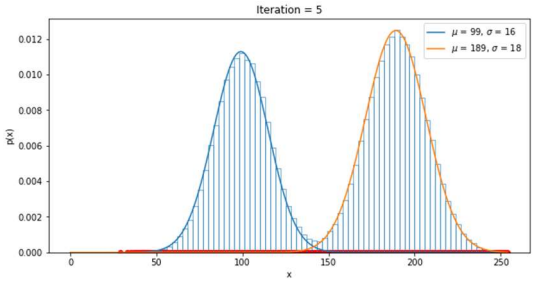
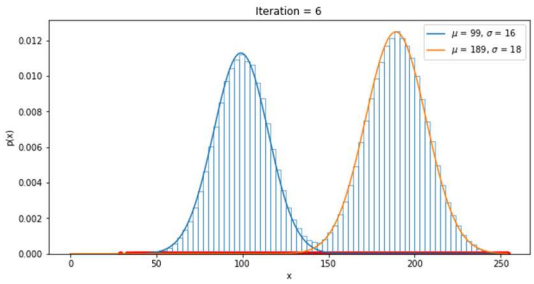
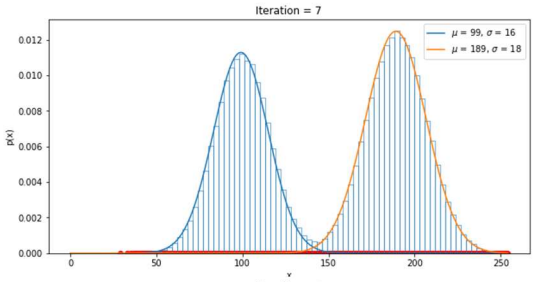
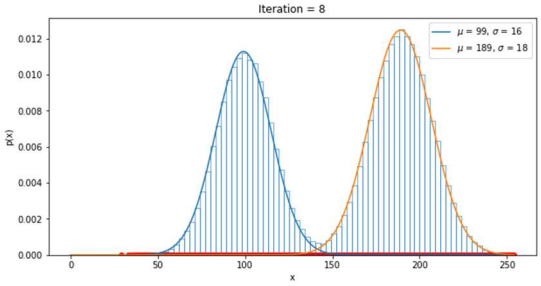


Figure 27: LCG EM on Lung: Iteration 0, 1, 2, 3, 4









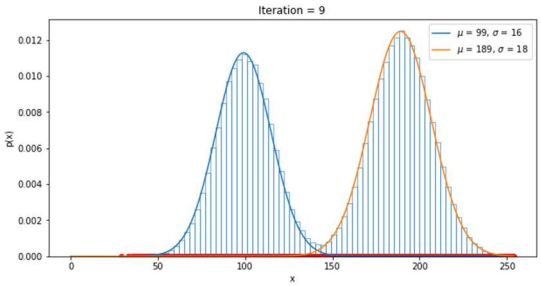
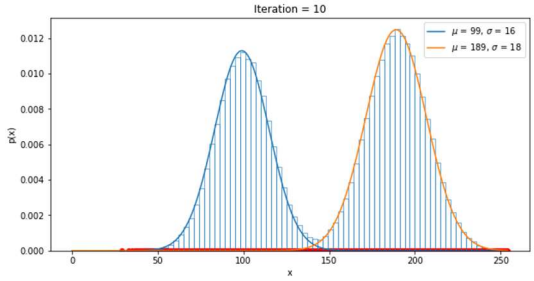
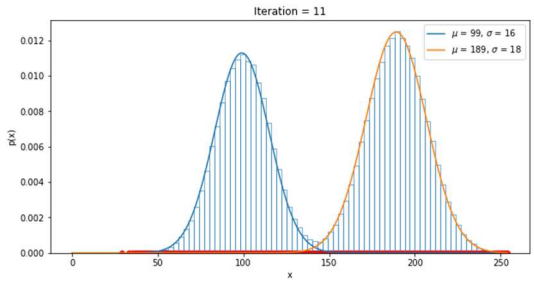
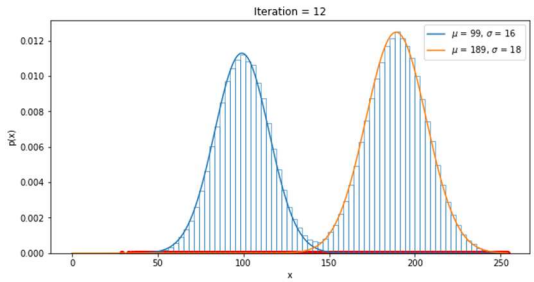
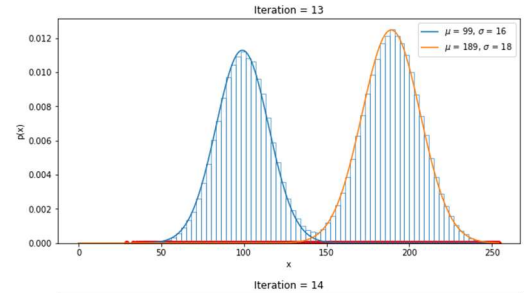


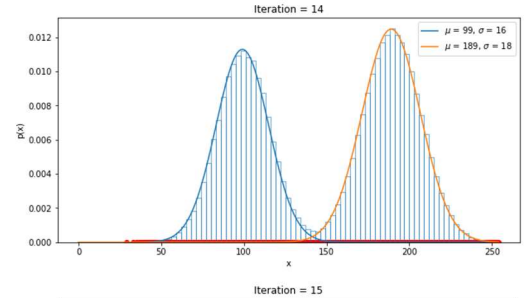
Figure 28: LCG EM on Lung: Iteration 5, 6, 7, 8, 9











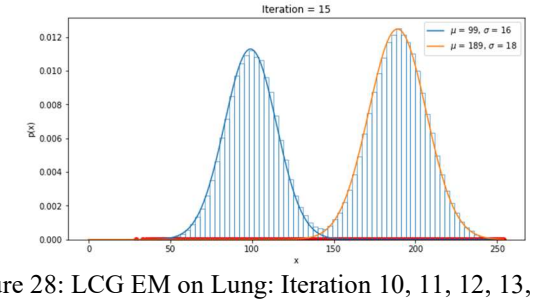


Figure 28: LCG EM on Lung: Iteration 10, 11, 12, 13, 14, 15

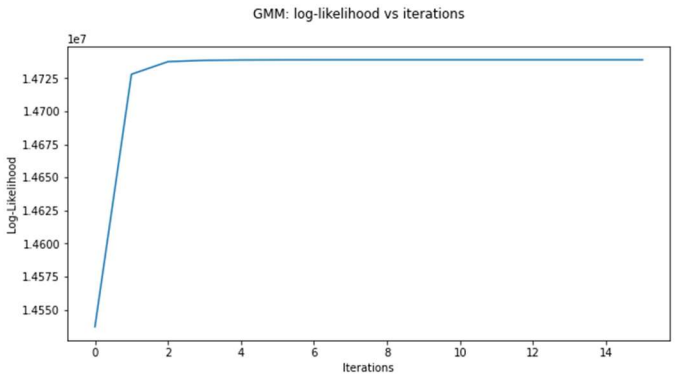


Figure 29: Log-likelihood vs. Iterations for Lung

After achieving a log-likelihood plot that showed very little change after iteration 16 we can conclude that the lung image gaussian density estimates were separated as early as iteration 2 but continued to have slight changes until iteration 6. The threshold was obtained from the EM algorithm from returning the distributions in a K by X.length array and replotted for finding the threshold. Since distribution array is set up as a K by X.length array we can use the X coordinate for which distribution K is less than distribution K – 1 (i.e. if K = 2 for the two gaussian curves in the lung image then threshold = j if distribution[K][j] < distribution[K-1][j] => distribution[2][141] < distribution[1][141] => threshold = 141. The X coordinate at 141 will be our threshold for the lung image as shown:

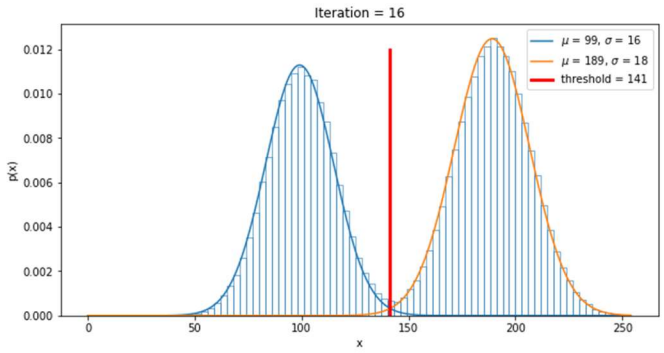


Figure 30: Intensity Threshold for Lung Image

And so the lung image vector may be segmented by setting all intensities greater than or equal to the threshold as an intensity of zero where as anything less than threshold = 141 will be set to 255. The segmented image vector is then reshaped using the length by width dimension reshaping used earlier to achieve the lung image vector and we achieve segmentation of the lungs:

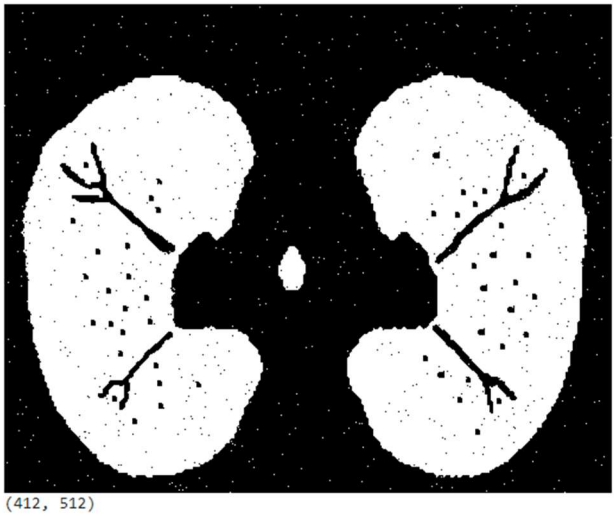


Figure 31: Segmentation of Lung Image

Now we move on to the kidney image. The same preprocessing was used for the kidney where we convert to a gray scale image, reshape to achieve a column vector, and plot the histogram. The main difference is that the kidney took a lot more iterations so by using a modulous of 10 we only plotted every 10 iterations of the EM algorithm progress as well as iteration zero so as not to use too much memory cause a major time complexity increase by storing and printing all these graphs or causing a stack overflow due to asking the graphics processing unit to store one hundred graphs when trying to tune the EM algorithm for this kidney image vector. The following kidney image vector management is similar in light to that of how we handled the lung image vector:

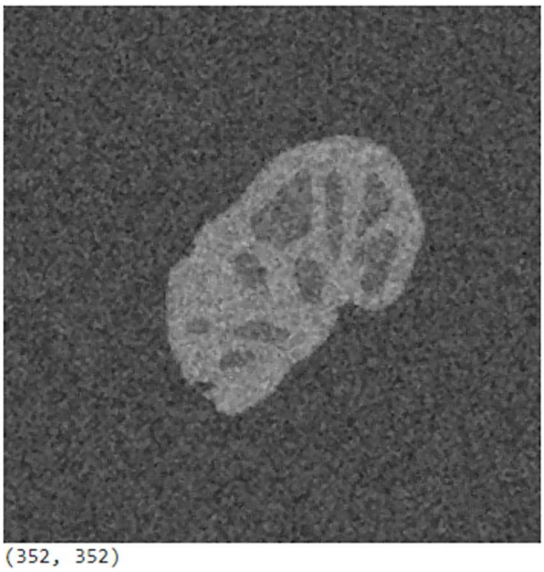


Figure 32: Imported Kidney Image

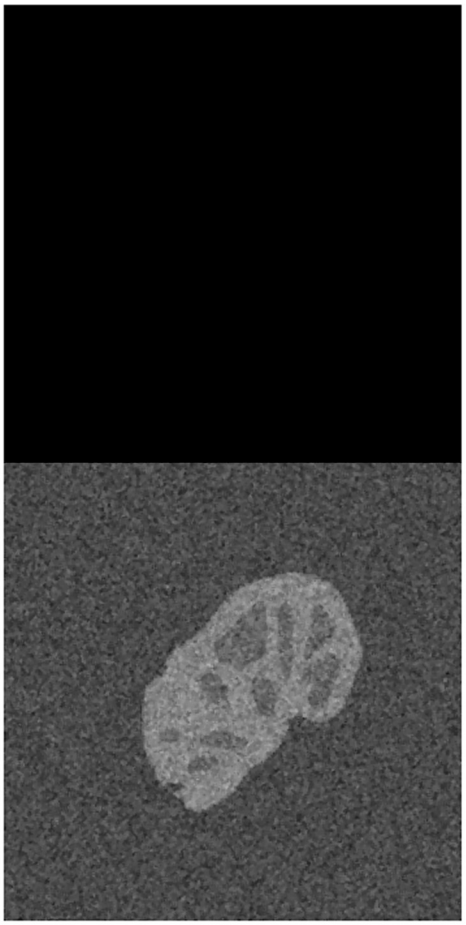


Figure 33: Normalized Kidney Image

```
[[ 0.]
 [79.]
 [56.]
 ...
 [70.]
 [42.]
 [ 0.]]
(123904, 1)
```

Figure 34: Kidney Image Vector Reshaping for application

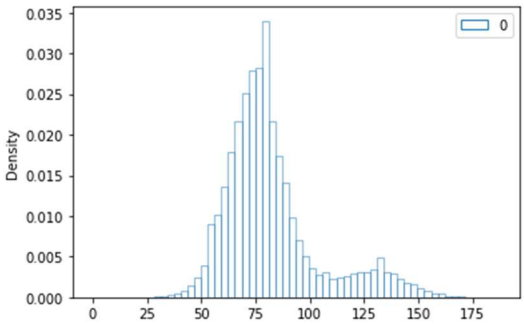


Figure 35: Kidney Image Vector Histogram

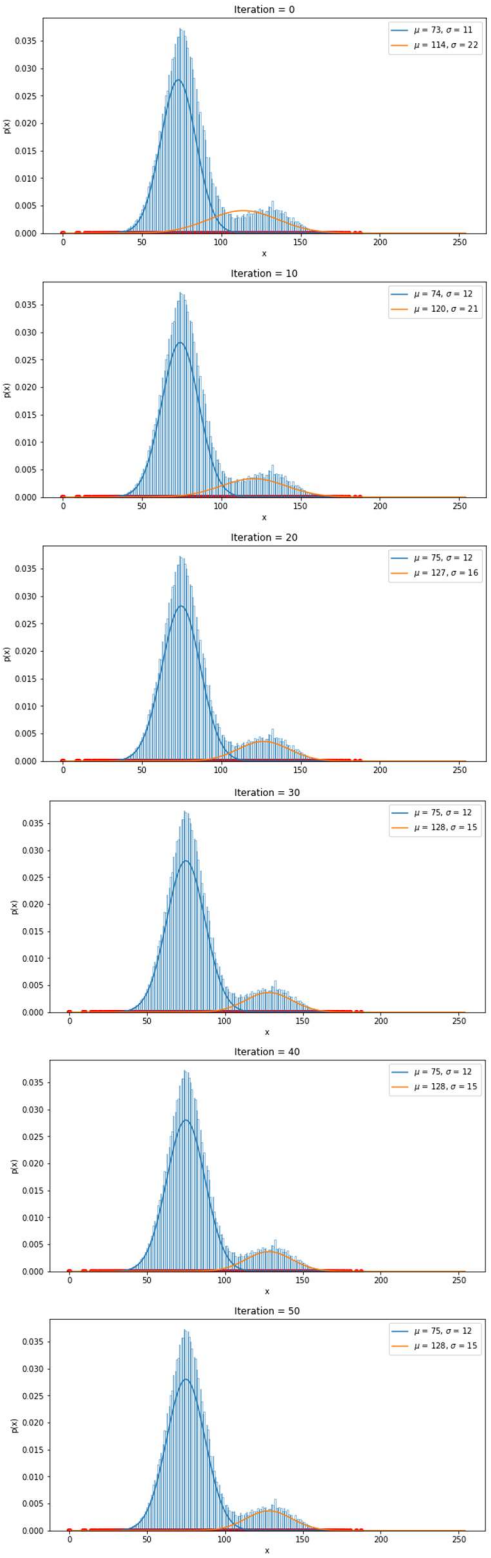


Figure 36: LCG EM on: Iteration 0, 10, 20, 30, 40 and 50

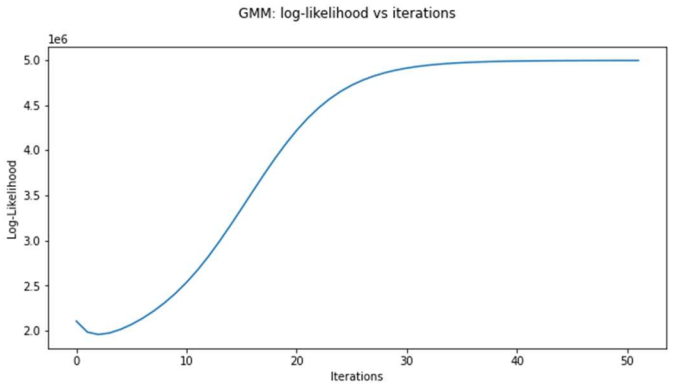


Figure 37: Log-likelihood vs. Iterations for Kidney

Since the log-likelihood vs iterations graph needs to be monotonically increasing; it might be worrisome that there is a slight dip at the start of the iterations, but the inaccuracy is negligible as will be seen when the threshold is calculated.

A similar approach for achieving the threshold was taken in the extraction of the distributions and threshold decision making for the distribution vectors extracted where we have a K by X.length vector where X has a range of our intensity levels 0 to 255. So, we decide all intensities below 105 will be set to an intensity of zero and above or equal to 105 will have an intensity of 255 as shown from the final iteration extracted from the EM algorithm:

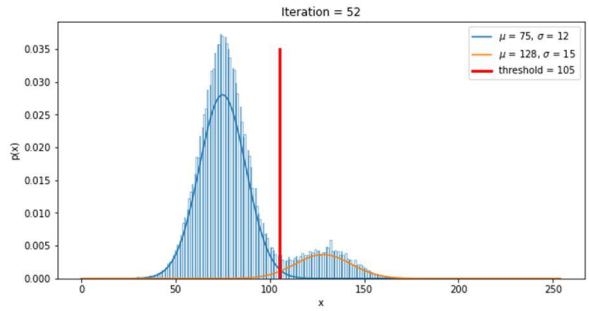


Figure 38: Intensity Threshold for Kidney Image

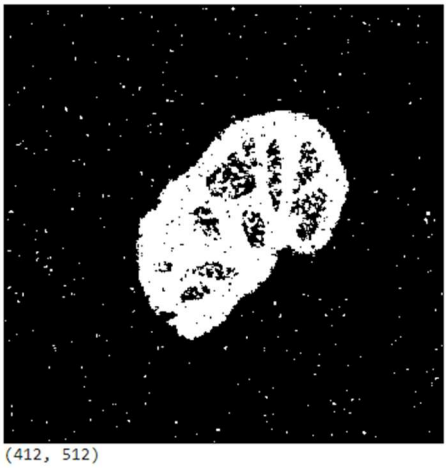


Figure 39: Segmentation of Kidney Image

II.a.4. PART 1.5: K-Component EM on Rose

Using the EM algorithm, estimate the marginal densities of the classes in the "rose.ppm" image. Get an initial segmentation based on these estimations.

The same procedure was taken when approaching the rose, except that the rose has an RGB channel depth for which the data will be much larger than the lung and kidney.

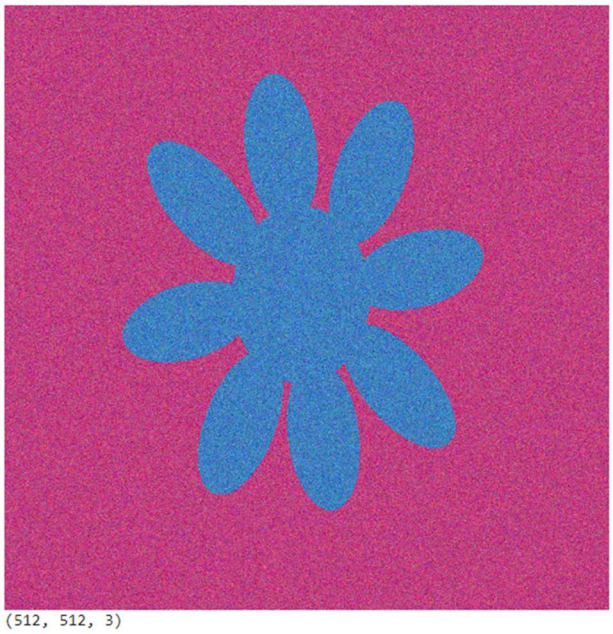


Figure 40: Imported Rose Image

```
[[136.]
 [ 73.]
 [222.]
 ...
 [146.]
 [ 80.]
 [179.]]
(786432, 1)
```

Figure 41: Rose Image Vector Reshaping for application

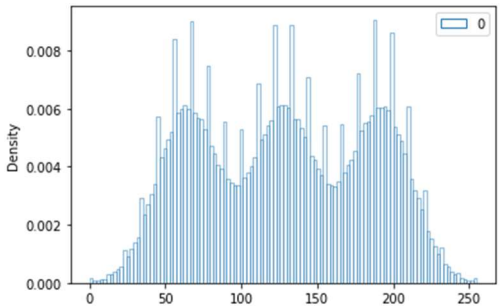


Figure 42: Rose Image Vector Histogram

After running the algorithm over night for 6 hours 47 minutes it was clear the data was too large for the implementation we used for the purpose of the project. The storage was overflowed and the time complexity was too large for 786 thousand data points from the rose image. However it should be stated if this ipython notebook were used in the presence of a better graphics processing unit and the implementation were more refined then we could achieve segmentation. If this is done segmentation should be performed by setting all intensities below 95 to zero (just by eye balling Figure 35 Histogram). Setting intensities between 95 and 170 to 127 and setting intensities above 170 to 255 will segment the image in the red, green, and blue channel. We could then reshape the image back into the Length, Width and Channel Depth of 512, 512, and 3, respectively.

Another approach is to convert the image to gray scale. However as shown, the image does not have clear edges in gray scale and the histogram loses the 2 gaussian curves from the 2 channel depths not used.

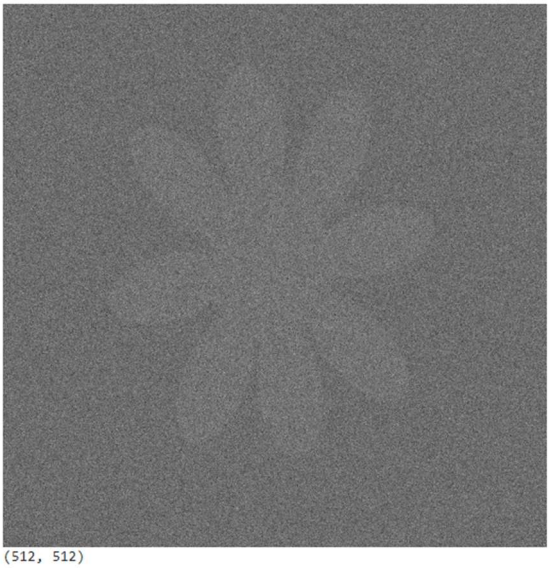


Figure 43: Gray Scale Rose Image

```
[[125.]
 [114.]
 [ 99.]
 ...
 [121.]
 [129.]
 [117.]]
(262144, 1)
```

Figure 44: Gray Scale Rose Image Vector Reshaping

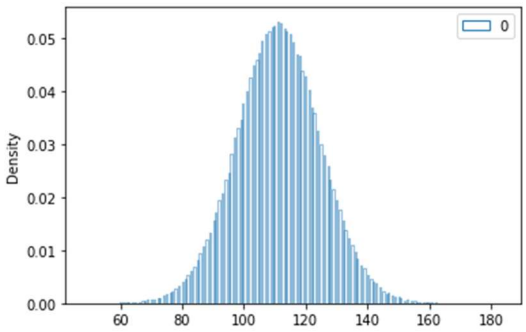


Figure 45: Gray Scale Rose Image Histogram

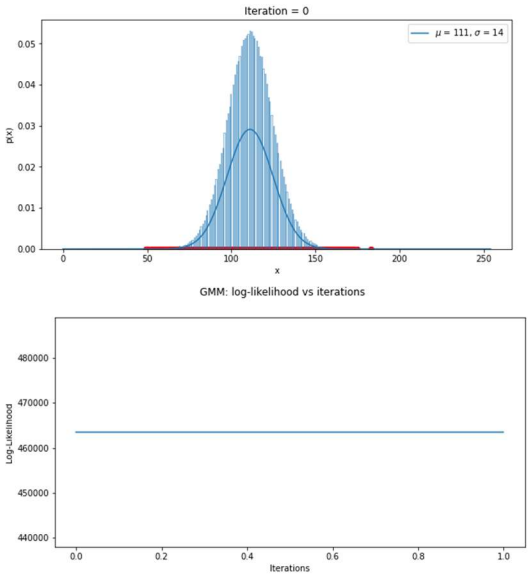
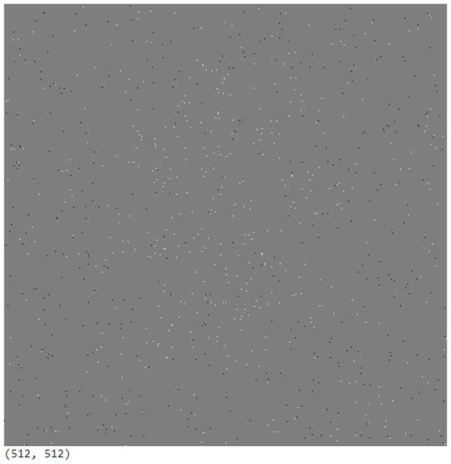


Figure 46: EM on Gray Scale Rose Image

It is clear from EM that anything below and above 3σ can be binary (i.e., minimum and maximum intensity) and anything between $\mu \pm 3\sigma$ should be set to a middle intensity of $\left(\frac{2^8}{2}\right) - 1 = \frac{256}{2} - 1 = 128 - 1 = 127$. So anything between $\mu - 3\sigma = 111 - 3(14) = 69$ and $\mu + 3\sigma = 111 + 3(14) = 153$. This means any intensity between 69 and 153 should be set to a gray scale intensity of 127 because the law of 6σ states anything in the range of $\mu \pm 3\sigma$ will resolve 99.7% of the data under a gaussian distribution.

```
segmented_rose_image = []
for i in range(0, len(rose_image_vector)):
  if (rose_image_vector.item(i) <= 69): # binary minimum segmentation
    segmented_rose_image.append(0)
  elif (rose_image_vector.item(i) > 69 and rose_image_vector.item(i) < 153):
    segmented_rose_image.append(127)
  else :
    segmented_rose_image.append(255)
segmented_rose_image = np.array(segmented_rose_image).reshape(Length, Width)

cv2_imshow(segmented_rose_image)
print(segmented_rose_image.shape)
```



(512, 512)

Figure 47: Gray Scale Rose Segmentation

It is clear that by doing this segmentation the noise in the grayscale image is too much without using the RGB channel depth necessary in order to achieve the edges of the rose image.

III. PART 2: DENSITY ESTIMATION & BAYESIAN CLASSIFICATION

The purpose of this experiment is twofold: Estimate a probability density function using non-parametric approaches (Parzen window and k-NN method) and use these estimates to implement Bayesian estimation.

III.a.1. PART 2.1: Generate Realizations:

First let us generate realizations from the following density functions:

a. Unit normal

b. A uniform density $p_1(x) = \begin{cases} \frac{1}{a} & if\ |x-b| \le \frac{a}{2} \\ 0 & otherwise \end{cases}$ for various a and b.

c. Triangular density $p_2(x) = \begin{cases} \frac{w+x-c}{w^2} & if\ c-w \le x \le c \\ \frac{w-x+c}{w^2} & if\ c \le x \le c+w \\ 0 & otherwise \end{cases}$ for various w and c.

d. $p(x) = \rho\, p_1(x) + \gamma\, p_2(x)$ as shown below, for various ρ and γ.

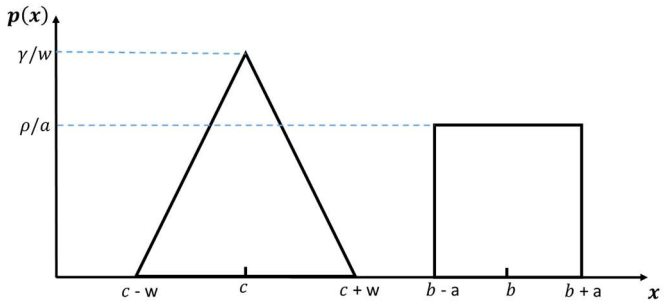


Figure 48: Density Plot of $p(x) = \rho\, p_1(x) + \gamma\, p_2(x)$

As produced for the gaussian mixture model we generate a standard uniform distribution, inverse transform the uniform distribution into a standard unit normal distribution and affine transform the standard normal distribution into a normal distribution with a desired mean and standard deviation.

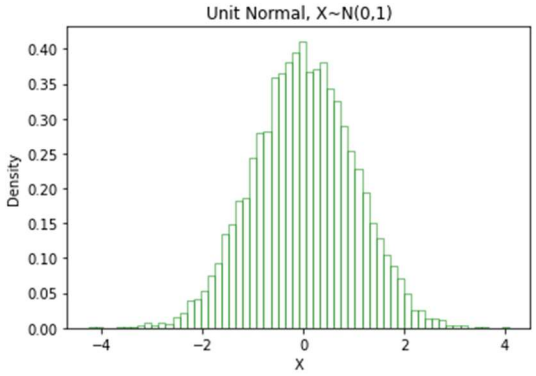


Figure 49: Unit Normal Distribution Generated Realization

The uniform distribution was set up to be generated between points a and b to generate the unit normal distribution so moving to the uniform distribution of any coordinate is done with ease.

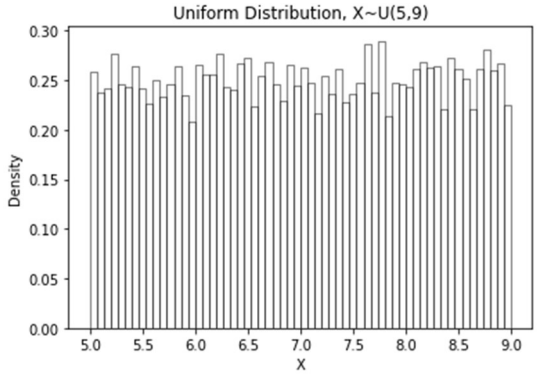


Figure 50: Uniform Distribution Generated Realization

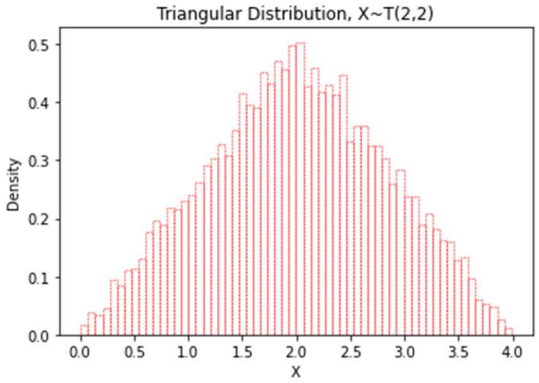


Figure 51: Triangular Distribution Generated Realization

The triangular distribution was generated with a newly designed inverse transform from the standard uniform distribution with a given center point of the triangle and a bandwidth for the triangle to span equidistant in the minor axis direction (i.e., the major axis is the density axis whereas the minor axis would be the points being generated and distributed.

A mixed distribution algorithm was designed to append the values of a generated triangular and uniform distributions to a list with an input scaling factor of each distribution ρ for p_1 or the uniform distribution and γ for p_2 or the triangular distribution. This means that the even indexed points will be triangular densities and the odd numbered indices will be uniform densities if the index starts at zero (i.e., zero is an even number and so i = 0, 2, 4, 6,… etc. will have triangular densities). This technique is similar to the Box-Muller transformation where the two smaller normal distributions are inter-spliced based on the odd and even indices.

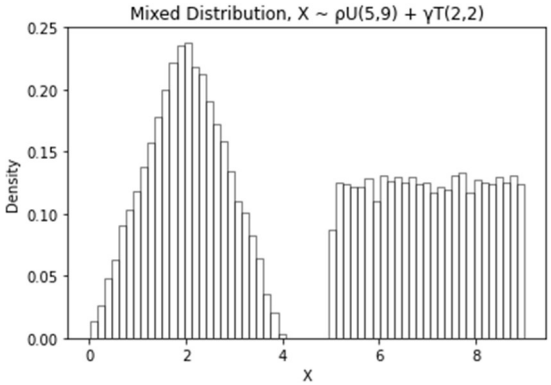


Figure 52: Mixed Distribution

III.a.2,3. PART 2.2 & PART 2.3: Parzen Window and K-NN on the Generated Realizations in Part 1:

Use the realizations in Problem 1 to estimate the density function using the Parzen window approach. Plot the nonparametric density estimates overlaid on the histogram of the data.

Use the realizations in Problem 1 to estimate the density function using the k-NN approach. Plot the nonparametric density estimates overlaid on the histogram of the data.

This was done simultaneously in that the Parzen window and K-NN could be compared on the same plot.

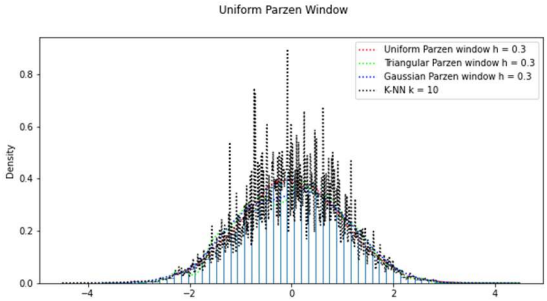


Figure 53: Parzen Window & K-NN on Unit Normal

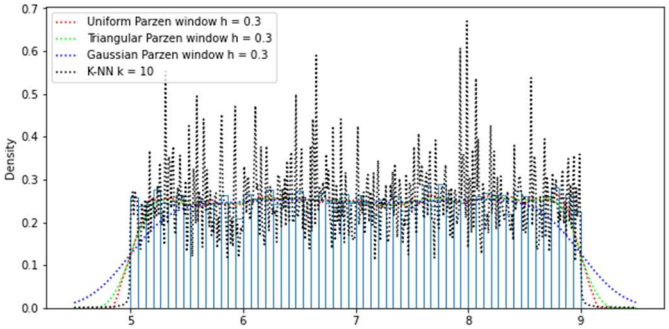


Figure 54: Parzen Window & K-NN on Uniform

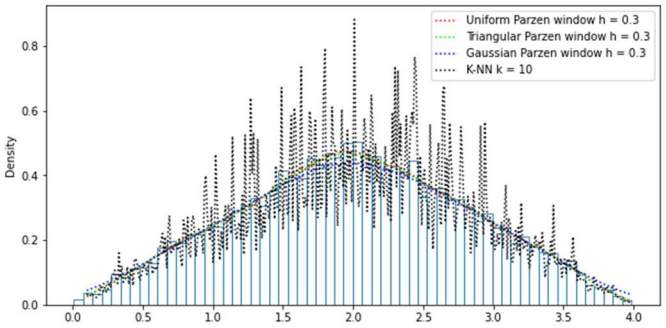


Figure 55: Parzen Window & K-NN on Triangular

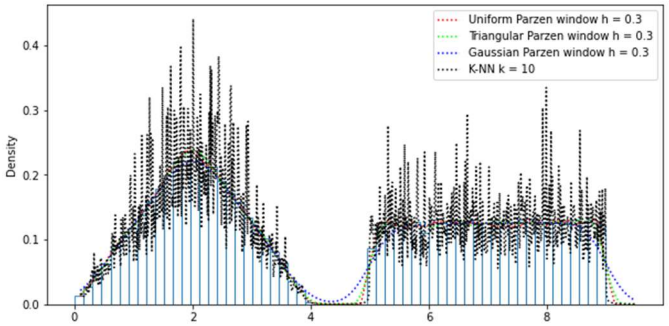


Figure 56: Parzen Window & K-NN on Mixed Distribution

The K-NN does not give the best density estimations, but still follows the trend of the distribution. In the following part we will explore the boundary conditions for the parzen window bandwidth and number of data points and the kernel k and number of data points for the K-NN non-parametric density estimates.

III.a.4. PART 2.4: The Parzen Window bandwidth and the K-NN kernel k

Use the realizations above to duplicate Figure 4.7 and 4.12 in Duda et al., 2001. Plot the nonparametric density estimates overlaid on the histogram of the data.

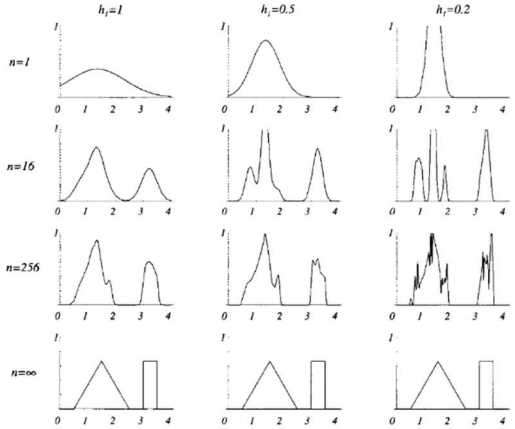


FIGURE 4.7. Parzen-window estimates of a bimodal distribution using different window widths and numbers of samples. Note particularly that the $n = \infty$ estimates are the same (and match the true distribution), regardless of window width.

Figure 57: Figure 4.7 from Duda, Hart, Stork

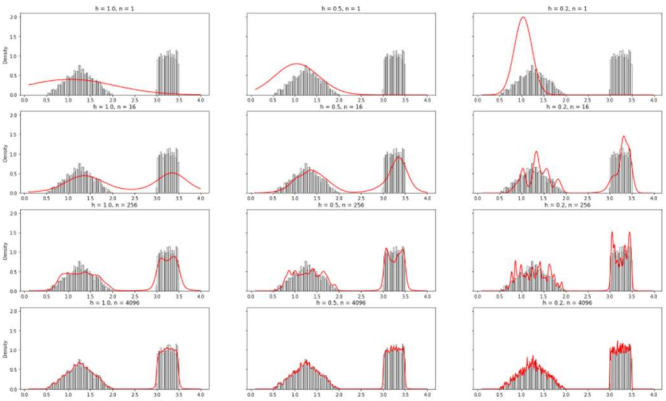


Figure 58: Replicating Figure 4.7

Some of the boundary conditions can be tricky during implementation. Either way we can see that as the number of data points increases the bandwidth no longer effects the density estimation and we get similar plots in the bottom row of the above Figure.

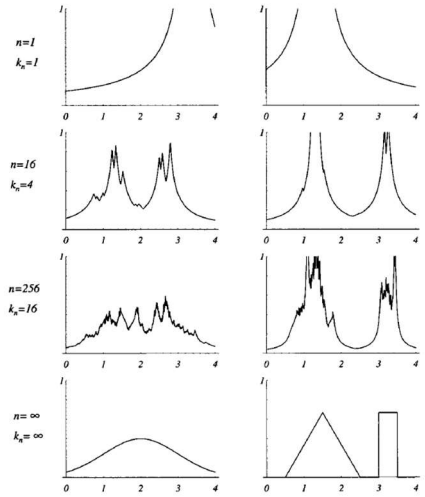


FIGURE 4.12. Several k-nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite "spiky."

Figure 59: Figure 4.12 from Duda, Hart, Stork

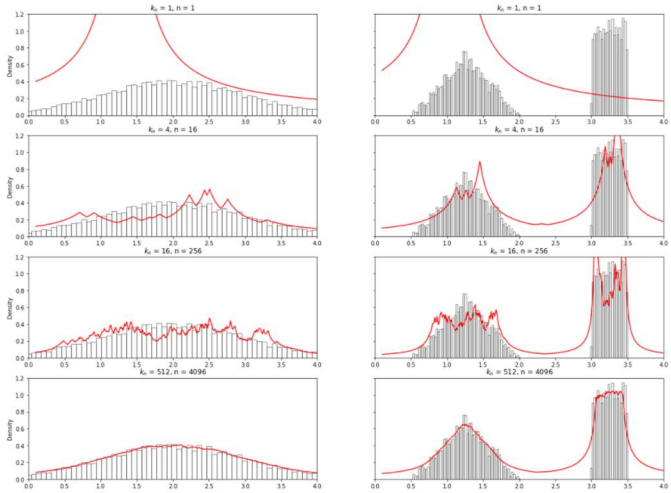


Figure 60: Replicating Figure 4.12

We can see the same in the above figure. As the number of data points increases the density estimation will almost perfectly estimate the density of the distributions for the gaussian and bimodal (i.e., mixed distribution).

III.a.5. PART 2.5: Variable number of K-components using EM algorithm on Generated Realizations

Use the realizations above to obtain the estimate of the density using the linear combination of Gaussian (LCG) approach with variable number of kernels in the EM algorithm.

Since we altered the generated realizations for the purpose of replicating the Figures from Duda, Hart and Stork we need to call the functions to regenerate the realizations and perform the EM algorithm on the distribution for various number of components gaussians.

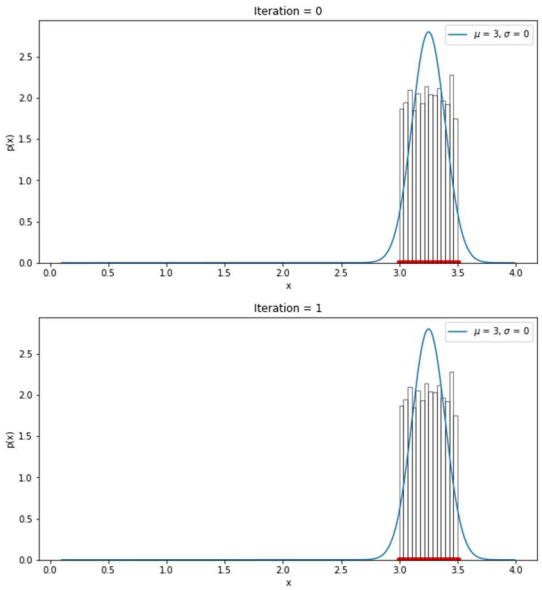


Figure 61: k = 1 EM Algorithm on Uniform

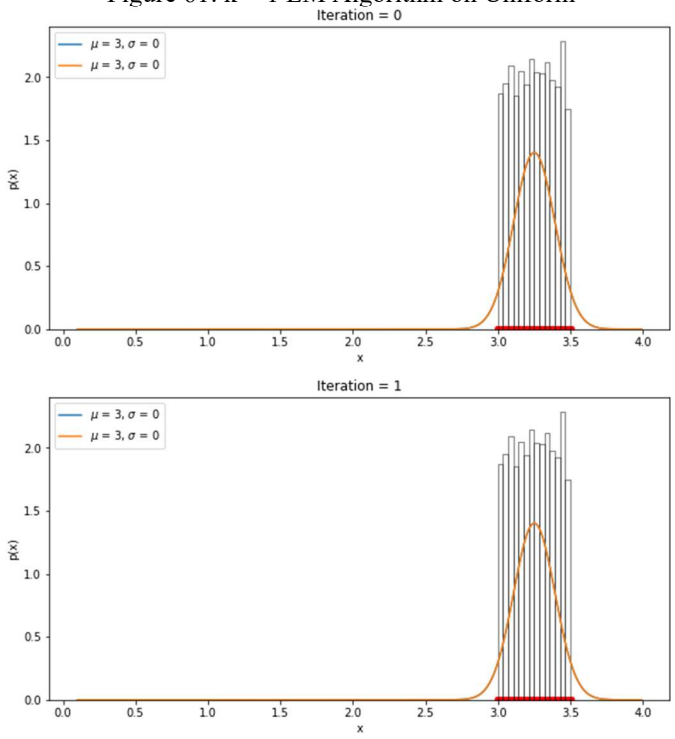


Figure 62: k = 2 EM Algorithm on Uniform

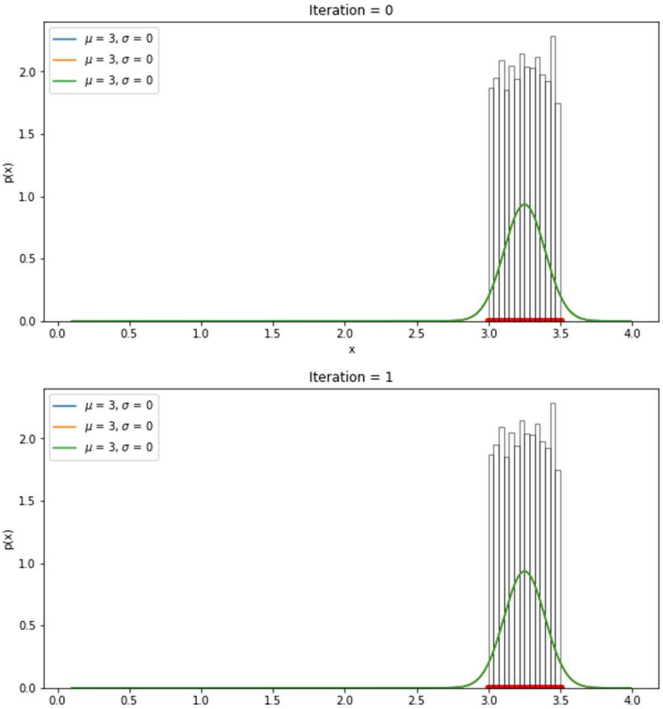


Figure 63: k = 3 EM Algorithm on Uniform

We can see form the above figures that as we add more components, each distribution area under the curve becomes smaller but equally distributed between the k components. So the total area under all the gaussian curves in each run of the EM algorithm will be equal.

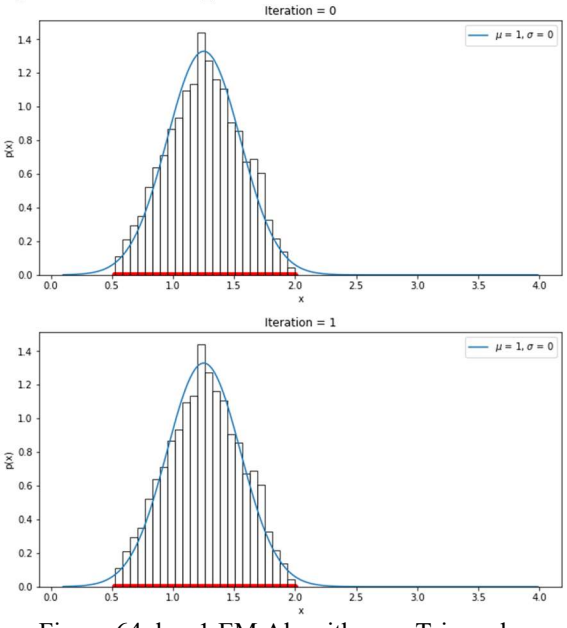


Figure 64: k = 1 EM Algorithm on Triangular

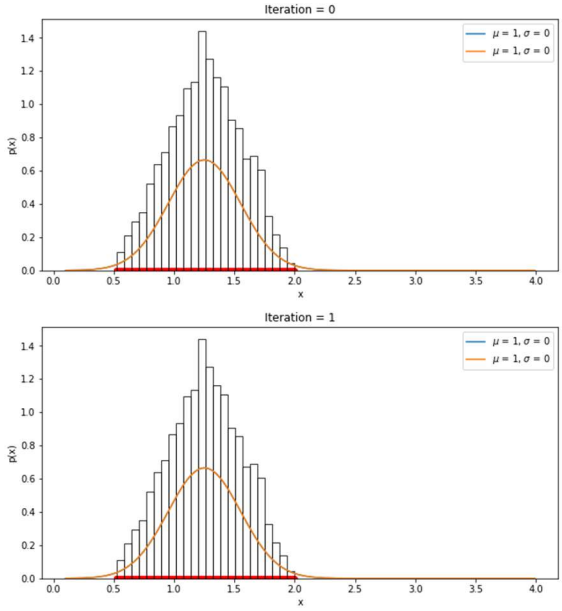


Figure 65: k = 2 EM Algorithm on Triangular

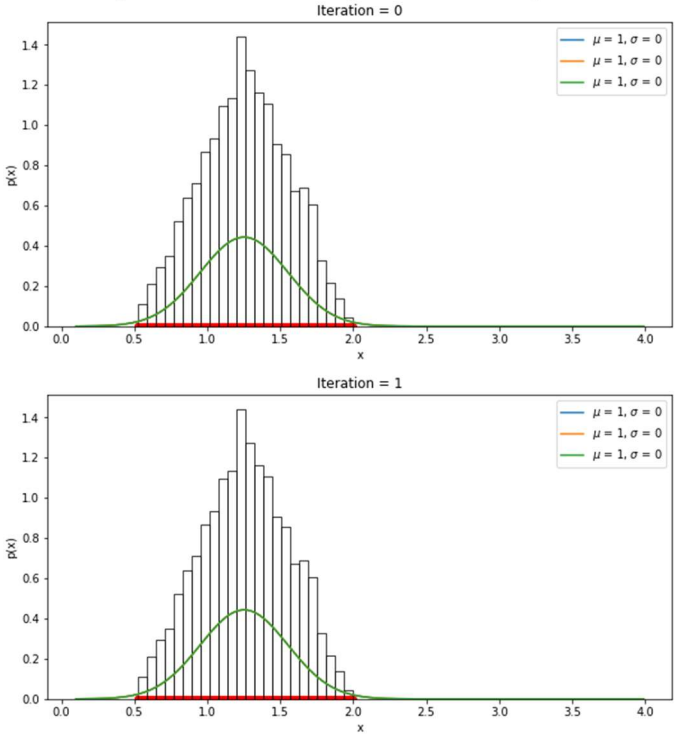


Figure 66: k = 3 EM Algorithm on Triangular

Again we see the equal k-component distributions overlap each other with equal area. For the Normal Distribution we will be using an affine transformed Normal Distribution with a mean of 2 and a standard deviation of 1 (i.e., X~N(2,1)).

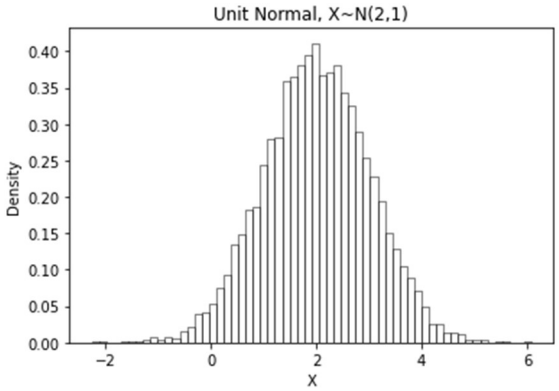


Figure 67: Generation of X~N(2, 1)

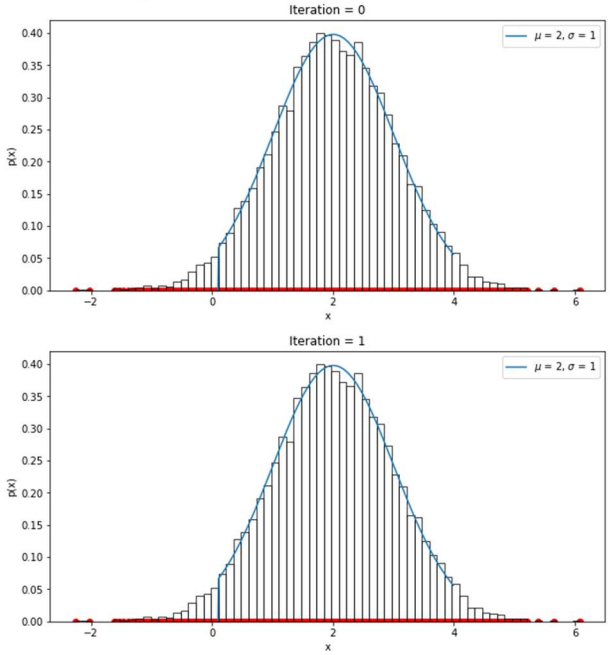


Figure 68: k = 1 EM Algorithm on affine Transformed Normal

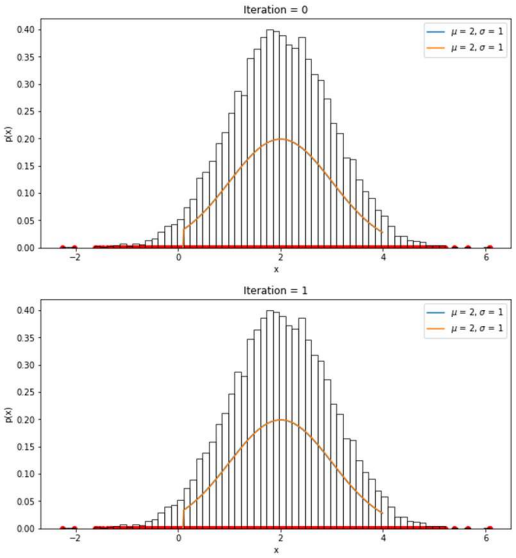


Figure 69: k = 2 EM Algorithm on affine Transformed Normal

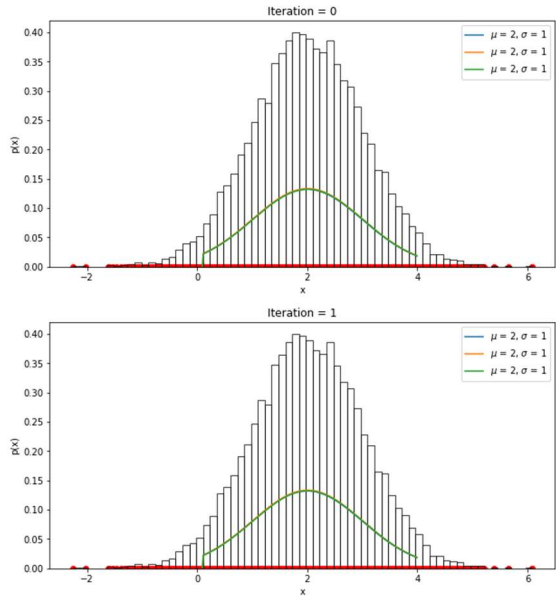


Figure 70: k = 3 EM Algorithm on affine Transformed Normal

Again we see similar results as in the uniform and triangular EM.

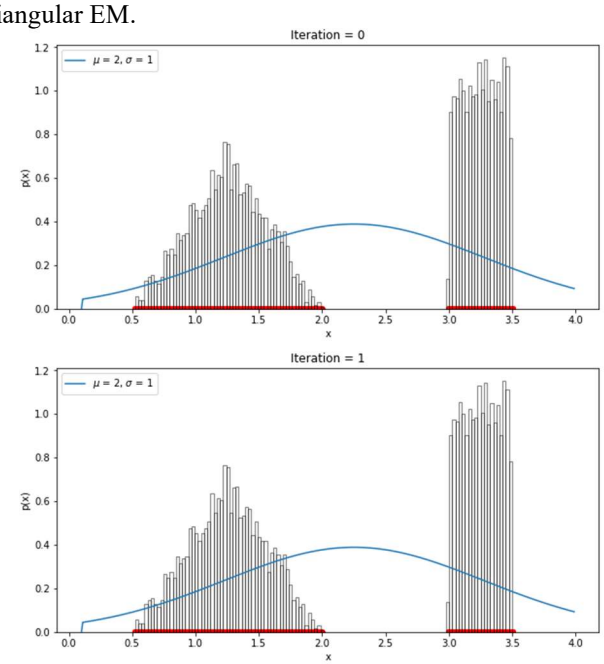


Figure 71: k = 1 EM Algorithm on Bimodal Mixed Distribution

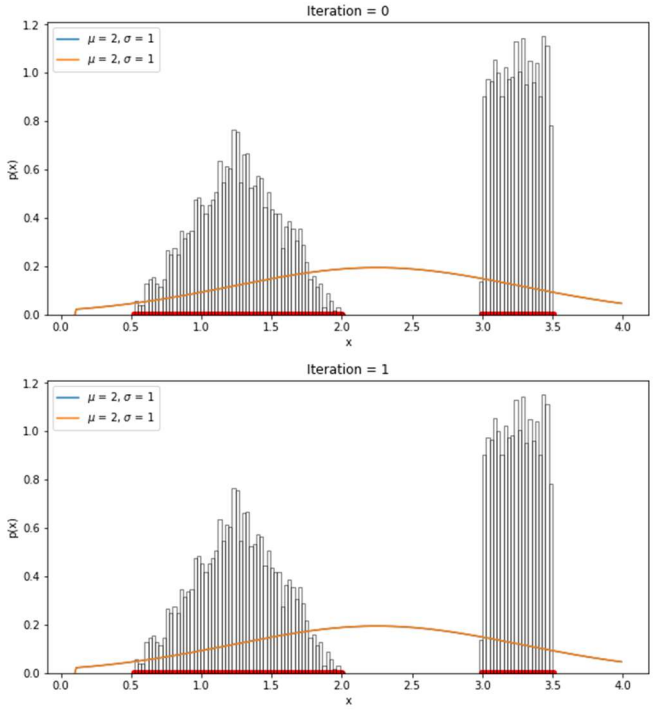


Figure 72: k = 2 EM Algorithm on Bimodal Mixed Distribution

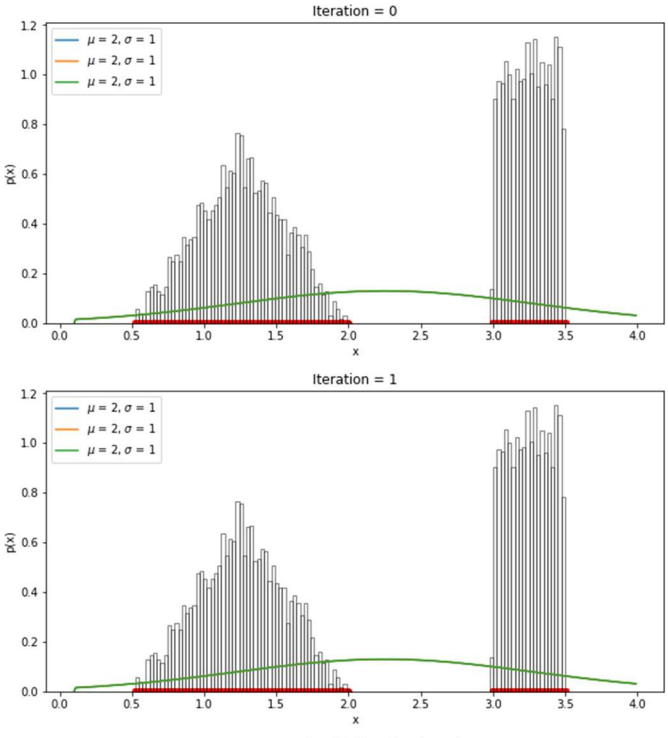


Figure 73: k = 3 EM Algorithm on Bimodal Mixed Distribution

From the above Figure there is something wrong with the implementation since the implementation we used is not recognizing the triangular and uniform distribution as two separate distributions for which the gaussians should be distributed. Possibly the weighting criteria could be altered for

each iteration so that the gaussian distribute themselves toward better recognized means from different distributions. But since the bimodal distribution is not a gaussian curve we will not focus on this implementation mistake.

III.a.6. PART 2.6: Bayesian classification of Generated Realizations:

Use the realizations in Problems 1-5 to design (i.e., devise the decision boundary) and test (i.e., devise the probability of error) a two-class Bayesian classifier. That is, class 1 is assumed to come from a unit Gaussian and class 2 comes from densities shown in Problems 1(b), 1(c) and 1(d), respectively.

For this we will generate 10,000 samples from each of the generated realizations and plot the density we have decided to use for the EM algorithm. We will then perform the EM algorithm on the generated density and see the results.

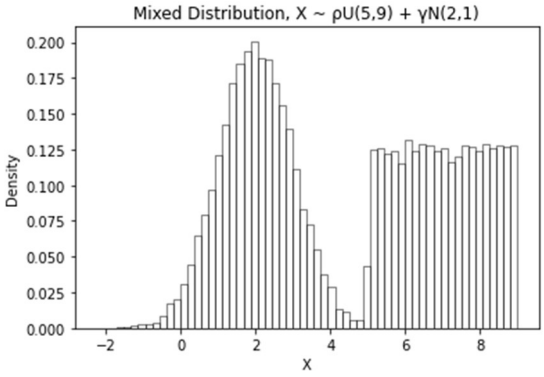


Figure 74: Generated Mixed Distribution

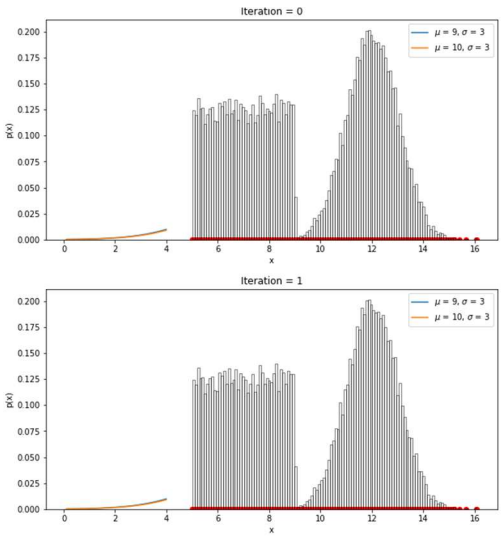


Figure 75: EM on Generated Mixed Distribution

It is clear there will need to be more implementation management of the EM algorithm to fine tune the inaccuracy of the code. The code seems to recognize gaussian curves but is having a hard time with anything besides a linear combination of gaussians (LCG). Again, weighted management of each gaussian should be the solution, but will be developed if needed for other applications.

IV. Conclusion

The aim of this series of projects was to implement density estimation using the nonparametric techniques Parzen Window and K-Nearest Nieghbor; the parametric technique Maximum Likelihood Estimation (MLE) and the LCG EM algorithm and use these implementations to segment images, and cluster data. It may be seen that the segmentation of images has produced good results. The clustering of data needs to be further explored and examined since the implementation of the EM algorithm did not produce the desired results. However, It is clear that given any number of combination of gaussian curves, this implementation can perform data clustering on that specific type of data set.

V. References

[1] Farag, Introduction to Probability Theory with Engineering Applications, Cognella Academic Publishing, 2021.

[2] Farag, Biomedical Image Analysis Statistical and Variational Approaches.

[3] Golub, Gene Howard, and Van Loan Charles F. *Matrix Computations*. The Johns Hopkins University Press, 2013.

[4] Reuven Rubinstein and Dirk P. Kroese Simulation and the Monte Carlo Method, 3rd Edition, Wiley, New Jersey, 2017

[5] R. O. Duda, P. E. Hart and D. G. Stork: Pattern Classification, 2nd Edition, Wiley, 2000.

[6] Spruyt, Vincent. "How to Draw an Error Ellipse Representing the Covariance Matrix?" *Computer Vision for Dummies*, 22 Mar. 2015, www.visiondummy.com/2014/04/draw-error-ellipse-representing-covariance-matrix/.

[7] Zurada, Jacek M. *Introduction to Artificial Neural Systems*. West Publishing Company, 1992

VI. Appendix: IPython Code:

The IPython Notebook is available at:
https://github.com/MichaelFerko