**Name - Niraj Kumar**

**ROll = 160455**

**Assignment 2**

In [ ]:

```
#importing all the necessary pacakgaes
import matplotlib.pyplot as plt
from sklearn import cluster, datasets, mixture
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.datasets import make_spd_matrix
plt.rcParams["axes.grid"] = False
```

In [ ]:

```
# define the number of samples to be drawn
n_samples = 100
```

In [3]:

```
# define the mean points for each of the systhetic cluster centers
t_means = [[8.4, 8.2], [1.4, 1.6], [2.4, 5.4], [6.4, 2.4]]

# for each cluster center, create a Positive semidefinite convariance matrix
t_covs = []
for s in range(len(t_means)):
  t_covs.append(make_spd_matrix(2))

X = []
for mean, cov in zip(t_means,t_covs):
  x = np.random.multivariate_normal(mean, cov, n_samples)
  X += list(x)

X = np.array(X)
np.random.shuffle(X)
print("Dataset shape:", X.shape)
```

```
Dataset shape: (400, 2)
```

In [4]:

```
# Create a grid for visualization purposes it is easy to visualize in this
x = np.linspace(np.min(X[...,0])-1,np.max(X[...,0])+1,100)
y = np.linspace(np.min(X[...,1])-1,np.max(X[...,1])+1,80)
X_,Y_ = np.meshgrid(x,y)
pos = np.array([X_.flatten(),Y_.flatten()]).T
print(pos.shape)
print(np.max(pos[...,1]))
```

```
(8000, 2)
11.625764766339984
```

In [5]:

```
# define the number of clusters to be learned since it was already given for 2 distribution mixtur
e model
# to differentiate from others I used 4 distribution gaussians for better visualization
k = 4

# create and initialize the cluster centers and the weight paramters
weights = np.ones((k)) / k # normalizing the weights
means = np.random.choice(X.flatten(), (k,X.shape[1])) # flattening to 1D
```

```
print(means)
print(weights)
```

```
[[5.5545612  5.33095217]
 [9.23480522 5.51556301]
 [5.24814228 0.49217443]
 [4.34353343 0.66447746]]
[0.25 0.25 0.25 0.25]
```

In [6]:

```
# create and initialize a Positive semidefinite convariance matrix as this will ensure the require
ment for derivatives
cov = []
for i in range(k):
  cov.append(make_spd_matrix(X.shape[1]))
cov = np.array(cov)
print(cov.shape)
```

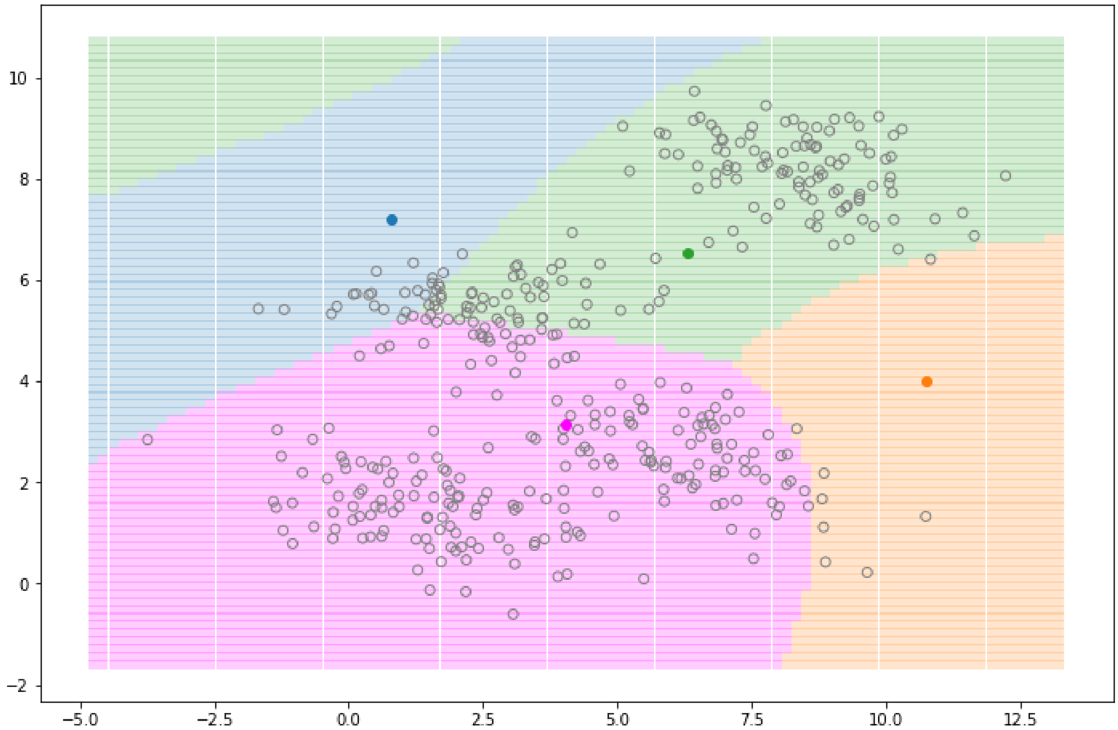
```
(4, 2, 2)
```

In [ ]:

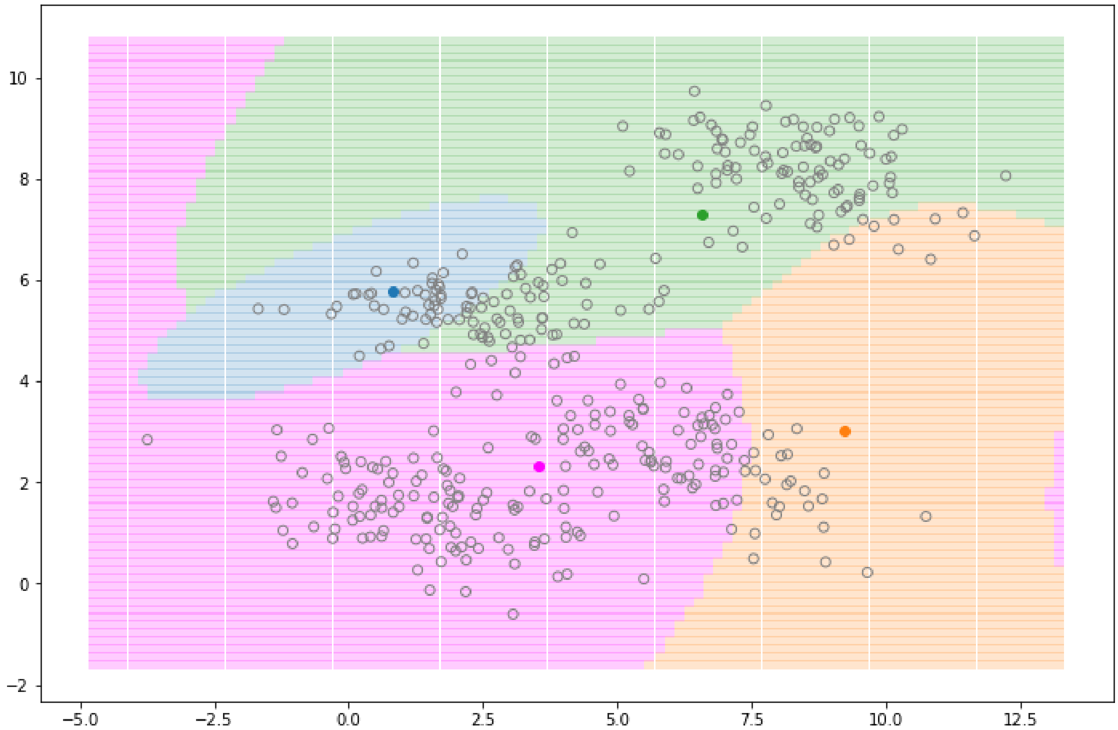
```
colors = ['tab:blue', 'tab:orange', 'tab:green', 'magenta', 'yellow', 'red', 'brown', 'grey'] # bet
ter vizualization of 4 Clusters
# since we used the 4 gaussians for this assignment so that will form 4 clusters
eps=1e-8

# run GMM for 40 steps
# we are running 40 steps random picked number
for step in range(40):

  # visualize the learned clusters
  if step % 1 == 0:
    plt.figure(figsize=(12,int(8)))
    plt.title("Iteration {}".format(step))
    axes = plt.gca()

    likelihood = [] # the respective likehood will be stored at each iteration after posterior has
been made into prior for next likelihood creation
    for j in range(k):
      likelihood.append(multivariate_normal.pdf(x=pos, mean=means[j], cov=cov[j]))
    likelihood = np.array(likelihood)
    predictions = np.argmax(likelihood, axis=0)

    for c in range(k):
      pred_ids = np.where(predictions == c)
      plt.scatter(pos[pred_ids[0],0], pos[pred_ids[0],1], color=colors[c], alpha=0.2, edgecolors='n
one', marker='s')

    plt.scatter(X[...,0], X[...,1], facecolors='none', edgecolors='grey')

    for j in range(k):
      plt.scatter(means[j][0], means[j][1], color=colors[j])


    plt.show()

  likelihood = []
  # Expectation step ( this will learn the posterior as the pi's are the priors of the distributio
ns using bayes theorem)
  for j in range(k):
    likelihood.append(multivariate_normal.pdf(x=X, mean=means[j], cov=cov[j]))
  likelihood = np.array(likelihood)
  assert likelihood.shape == (k, len(X))

  b = []
  # Maximization step (also known as M step ) this will try to find the mu,pi's,and covarince for
our clusters using the latent posterior
    # that we calculated form E step of the algorithm
  for j in range(k):
    # use the current values for the parameters to evaluate the posterior
    # probabilities of the data to have been generanted by each gaussian
    b.append((likelihood[j] * weights[j]) / (np.sum([likelihood[i] * weights[i] for i in range(k)],
```

```
axis=0)+eps))

    # updage mean and variance
    means[j] = np.sum(b[j].reshape(len(X),1) * X, axis=0) / (np.sum(b[j]+eps))
    cov[j] = np.dot((b[j].reshape(len(X),1) * (X - means[j])).T, (X - means[j])) / (np.sum(b[j])+eps
)

    # update the weights
    weights[j] = np.mean(b[j])

    assert cov.shape == (k, X.shape[1], X.shape[1]) # checking if the required dimensions are prese
nt or not
    assert means.shape == (k, X.shape[1])
```

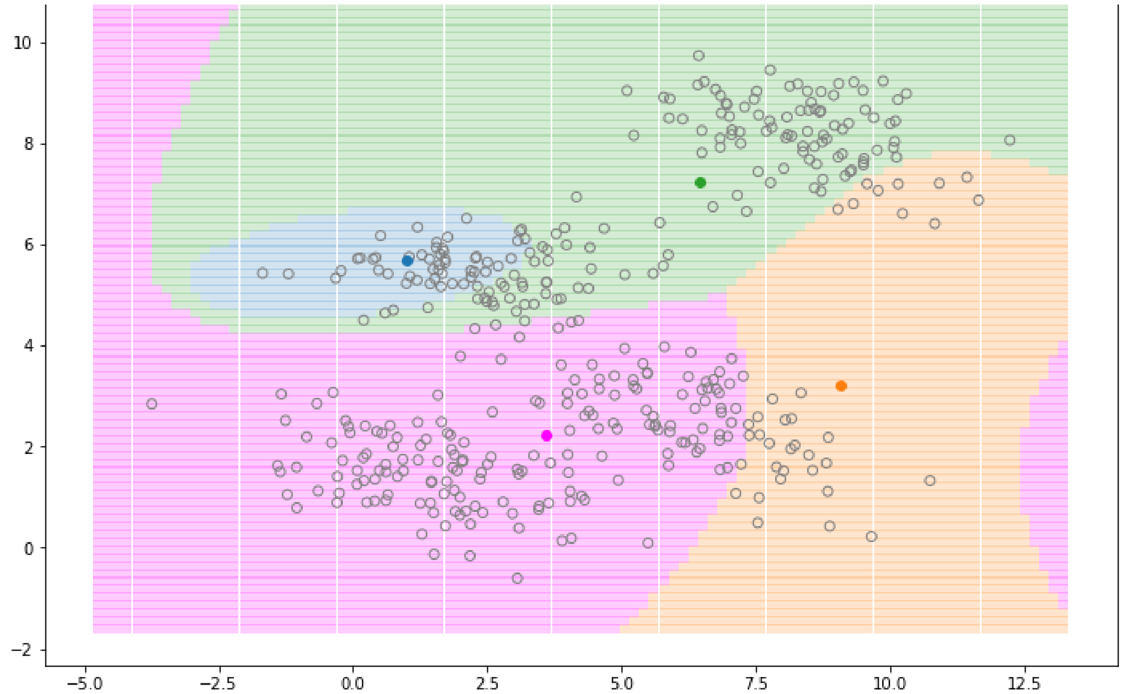
Iteration 0



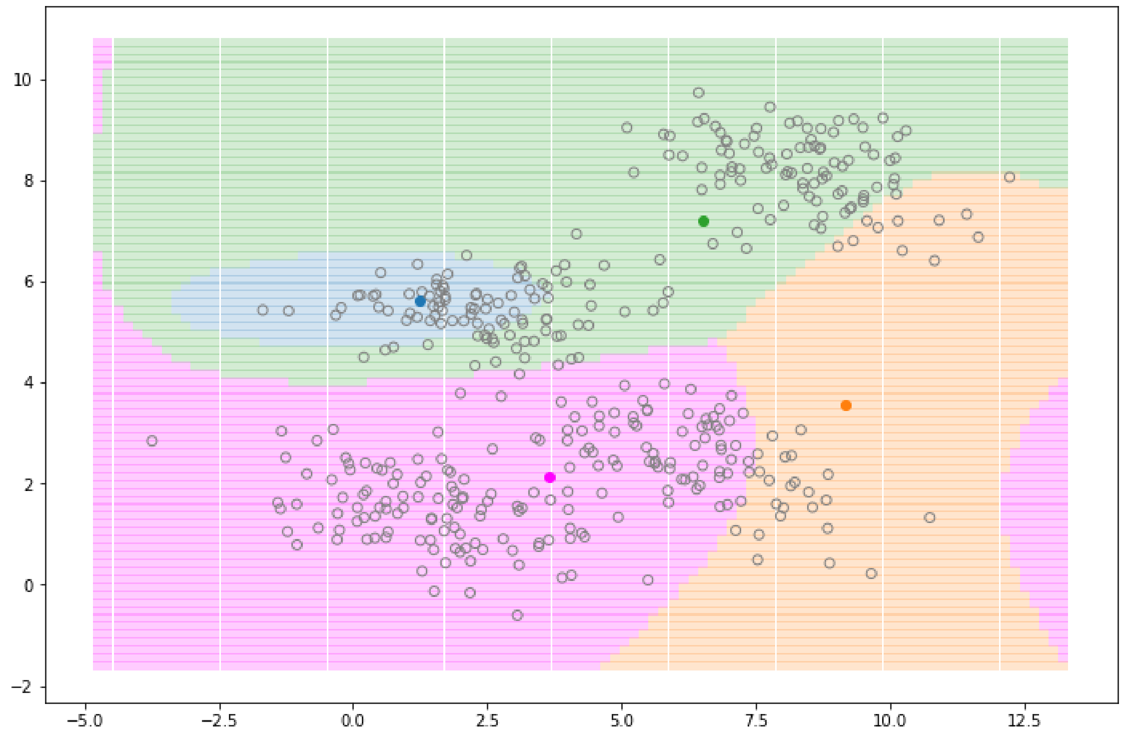

Iteration 1



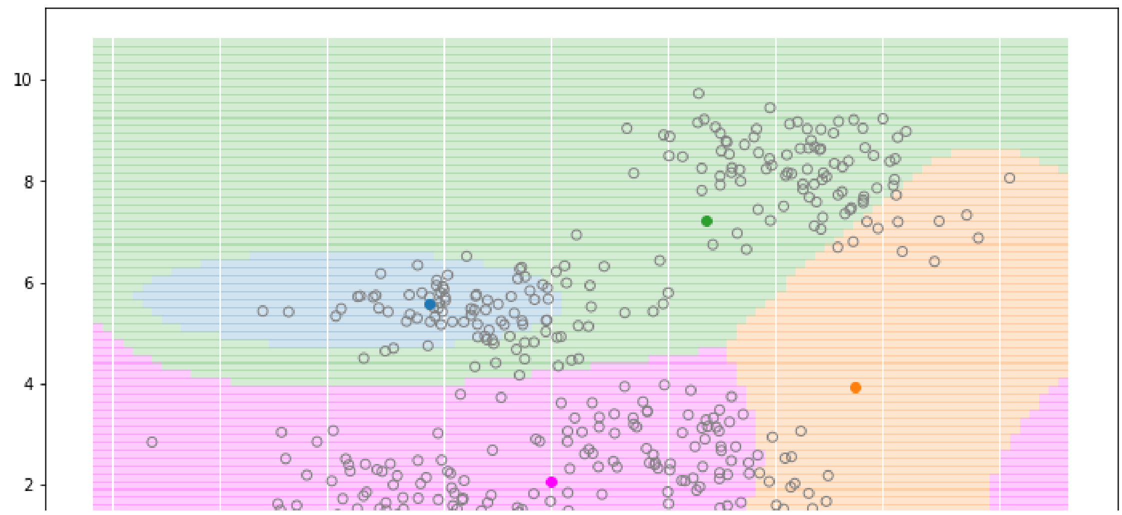

Iteration 2

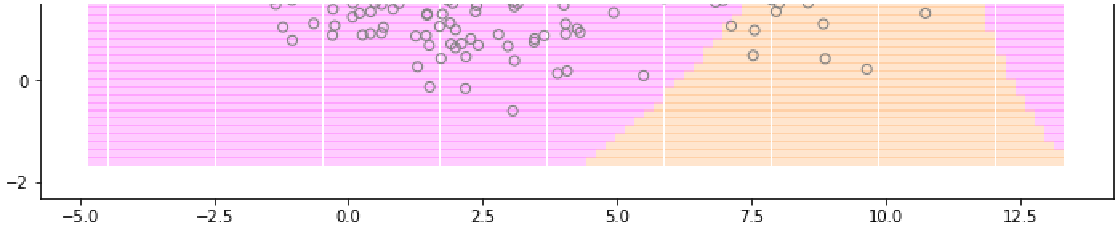


Iteration 3

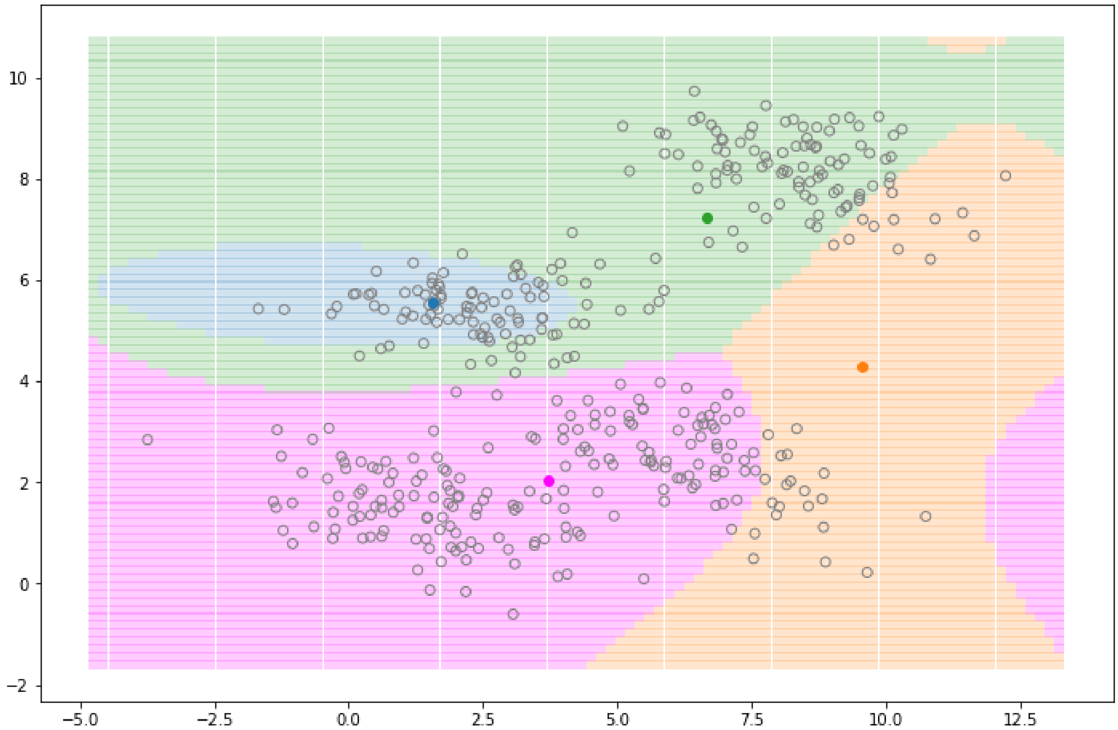


Iteration 4

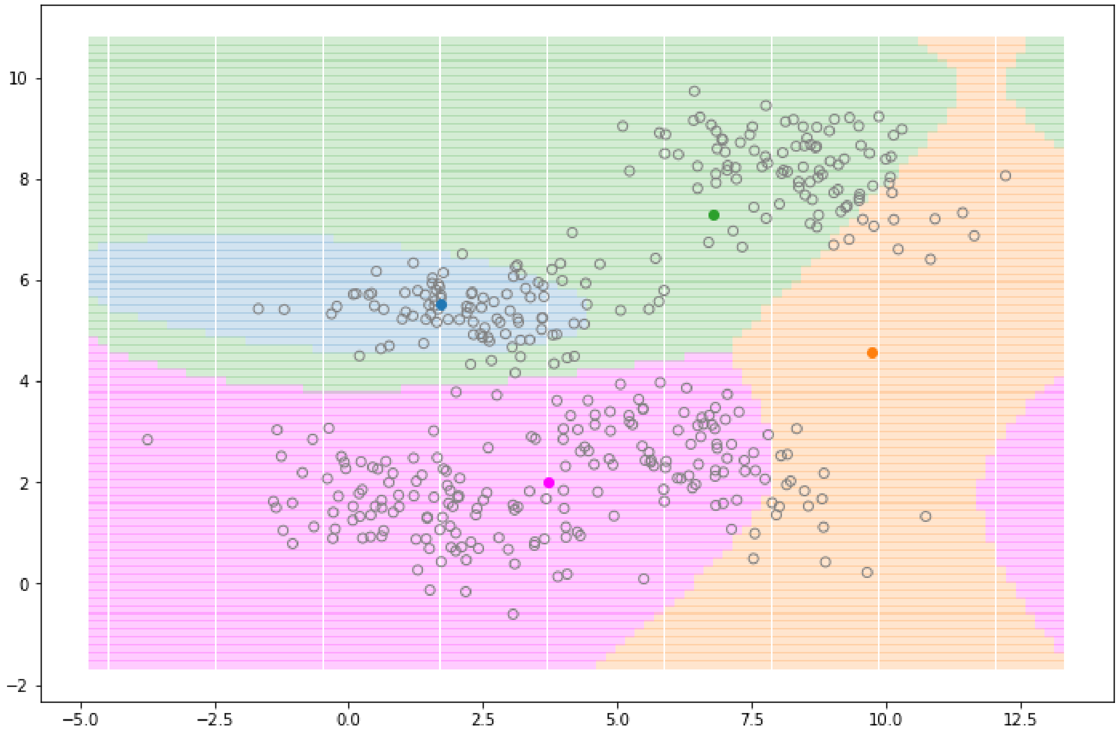




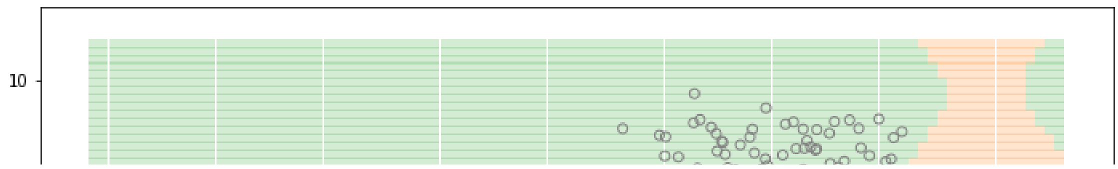

Iteration 5

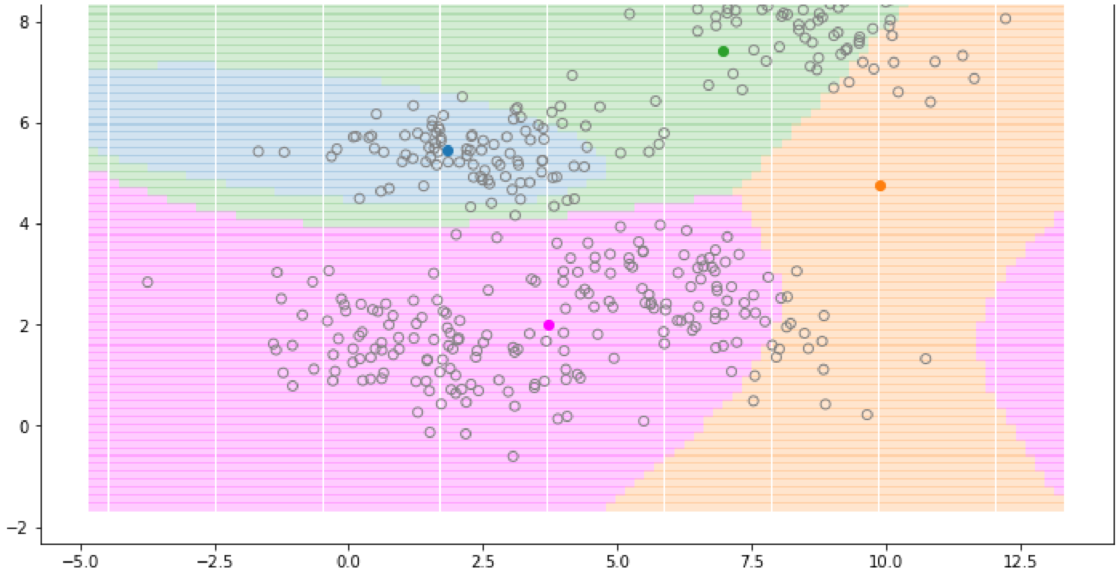



Iteration 6

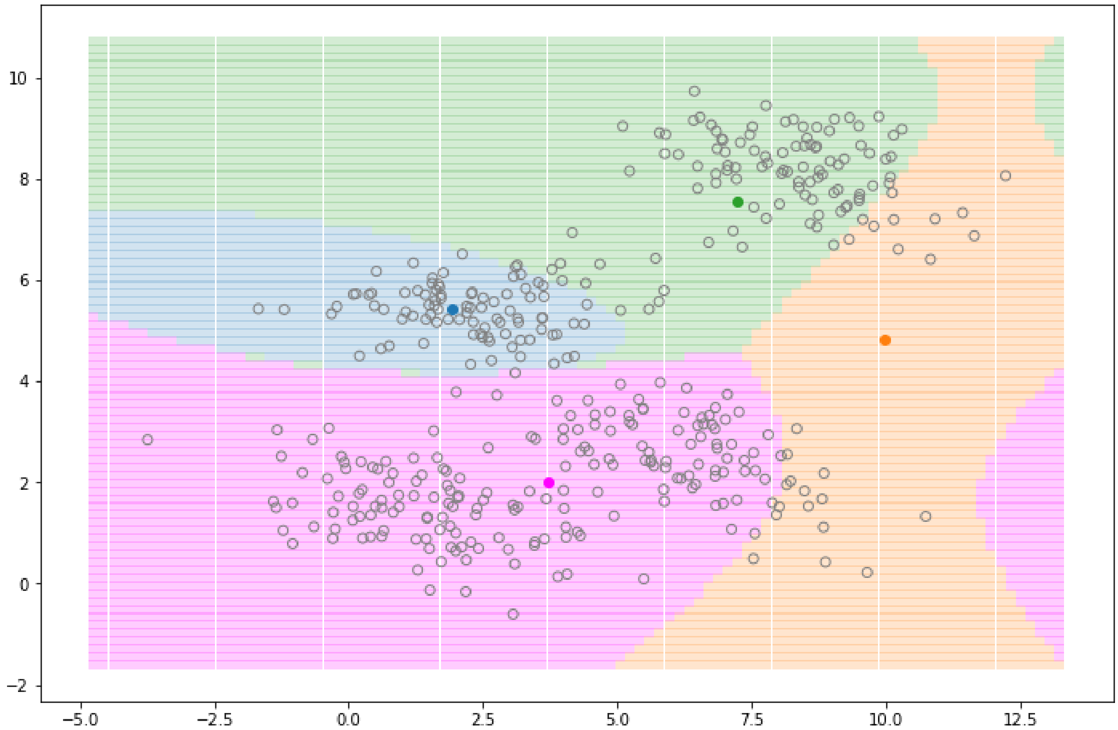



Iteration 7

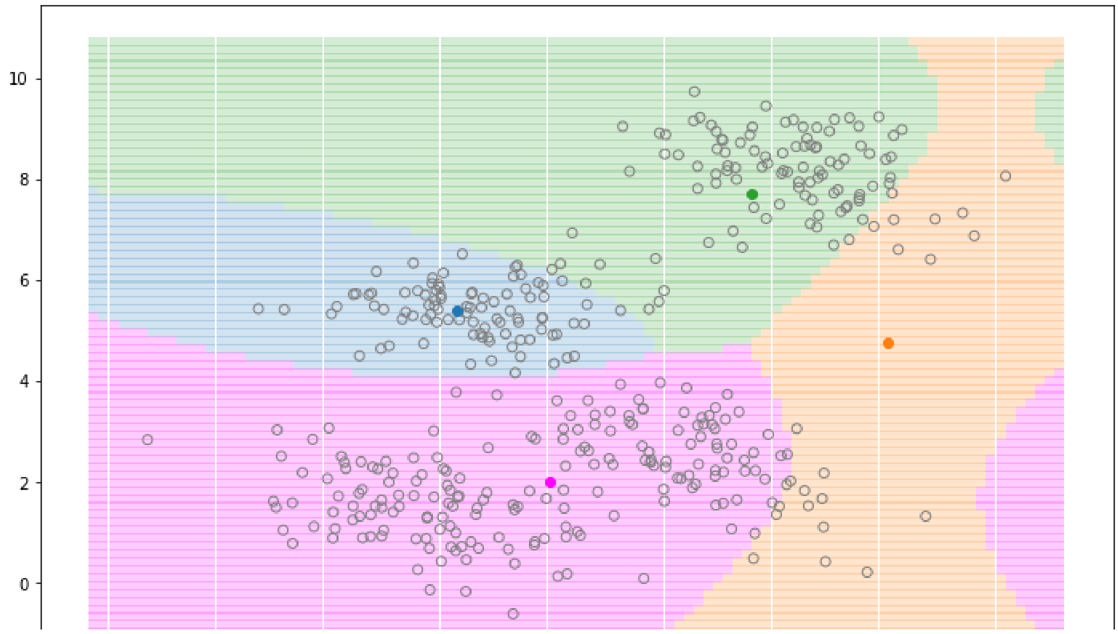


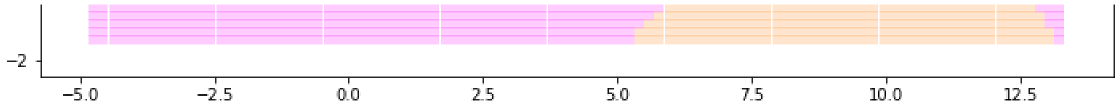



Iteration 8

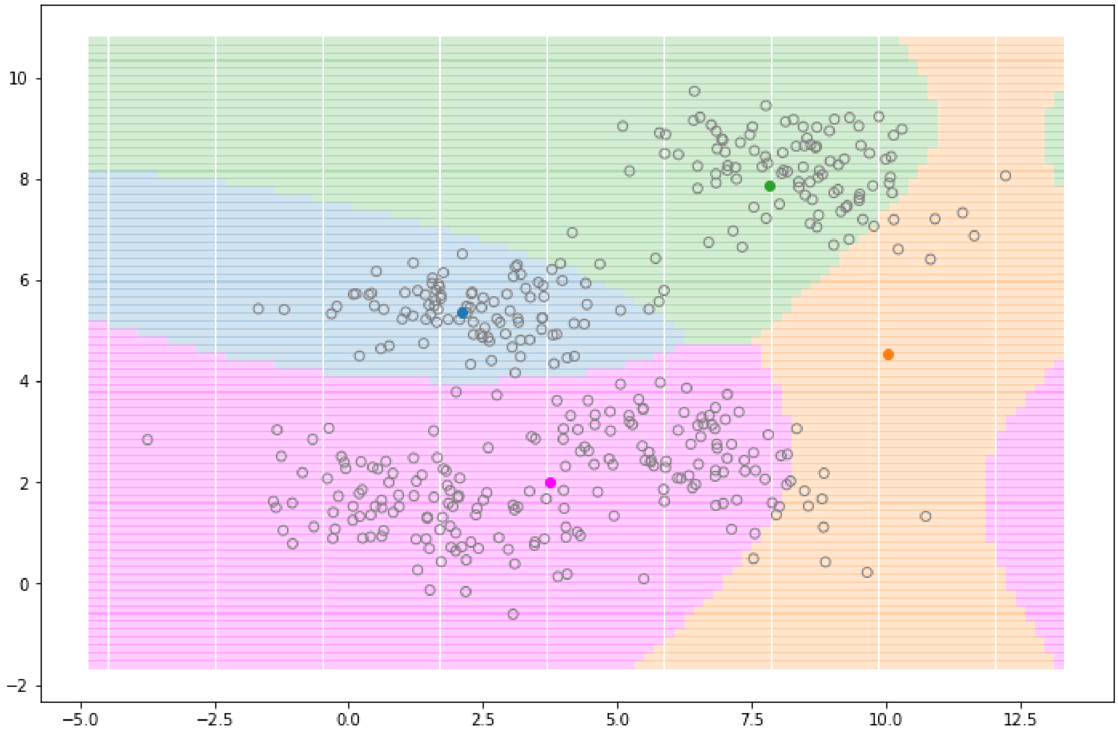



Iteration 9

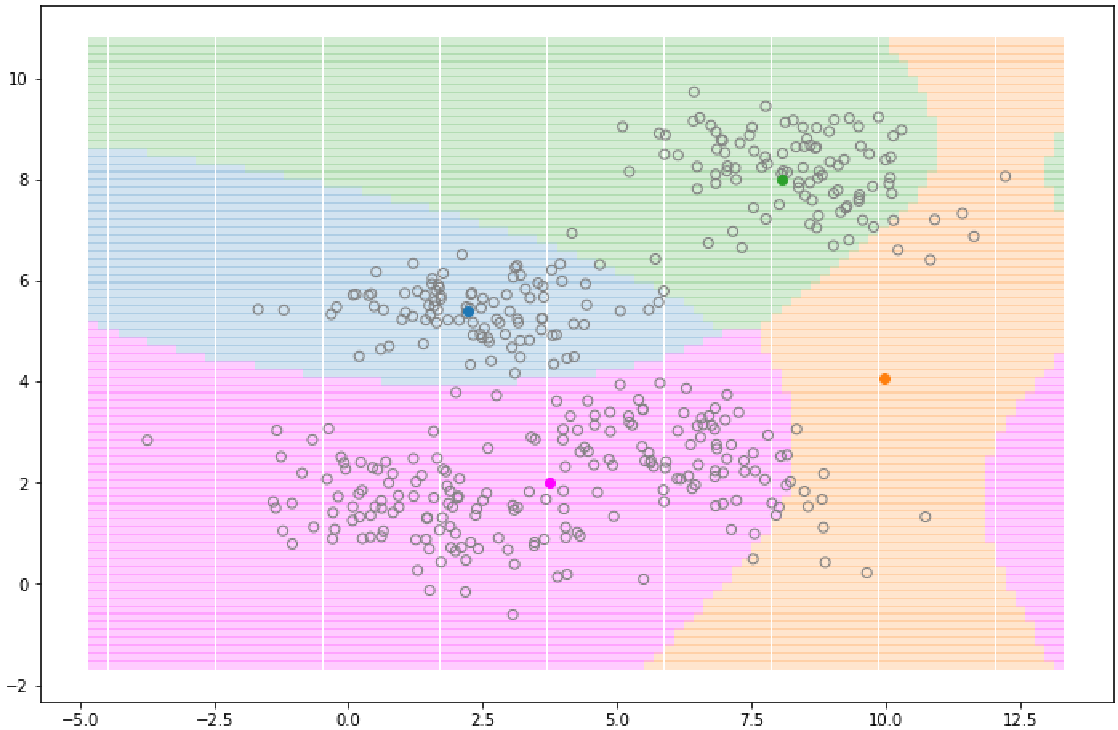




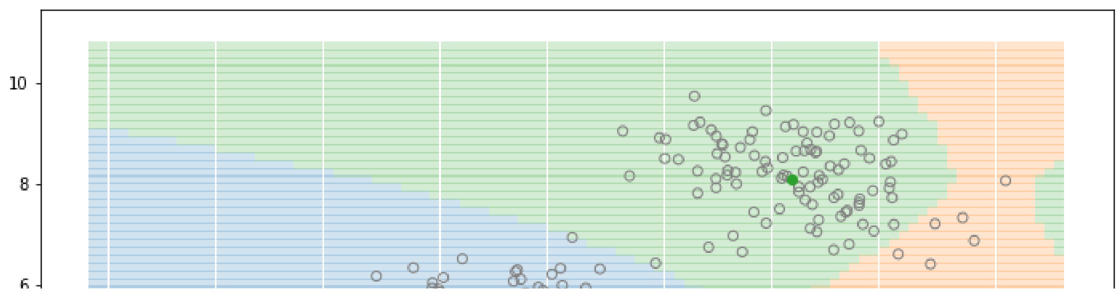

Iteration 10

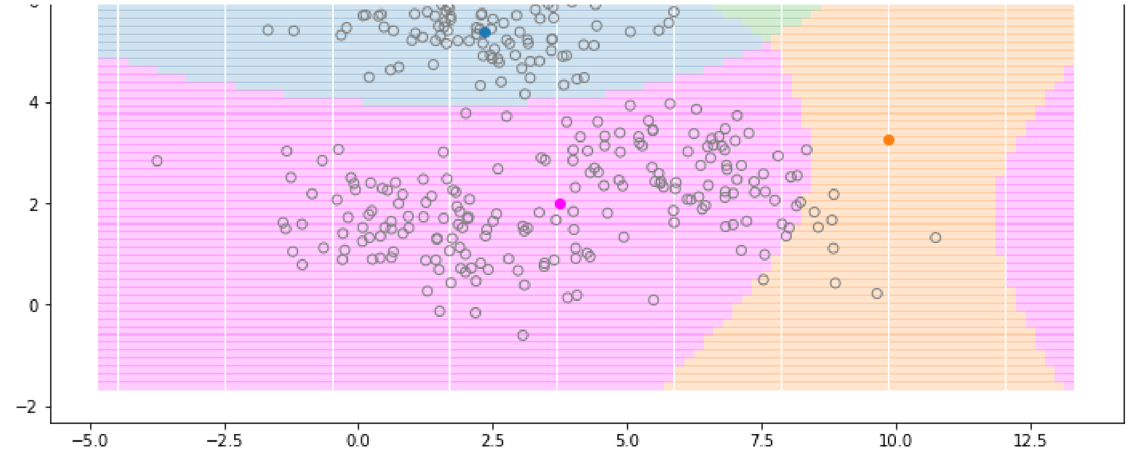



Iteration 11

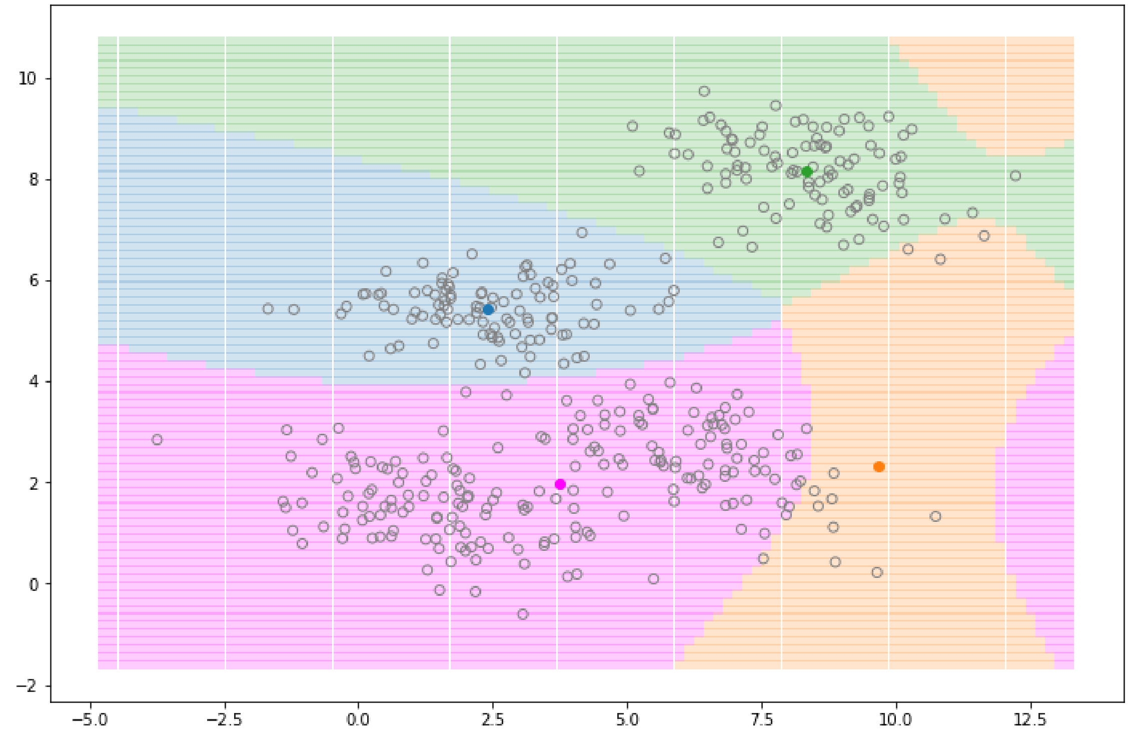



Iteration 12

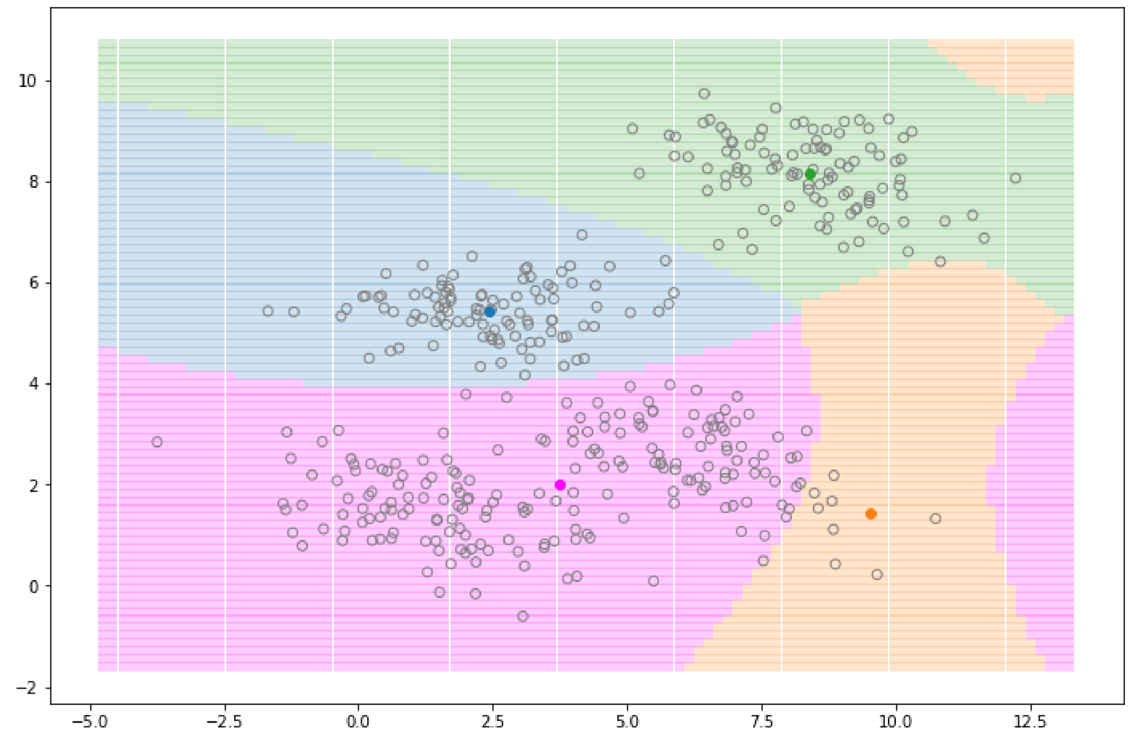




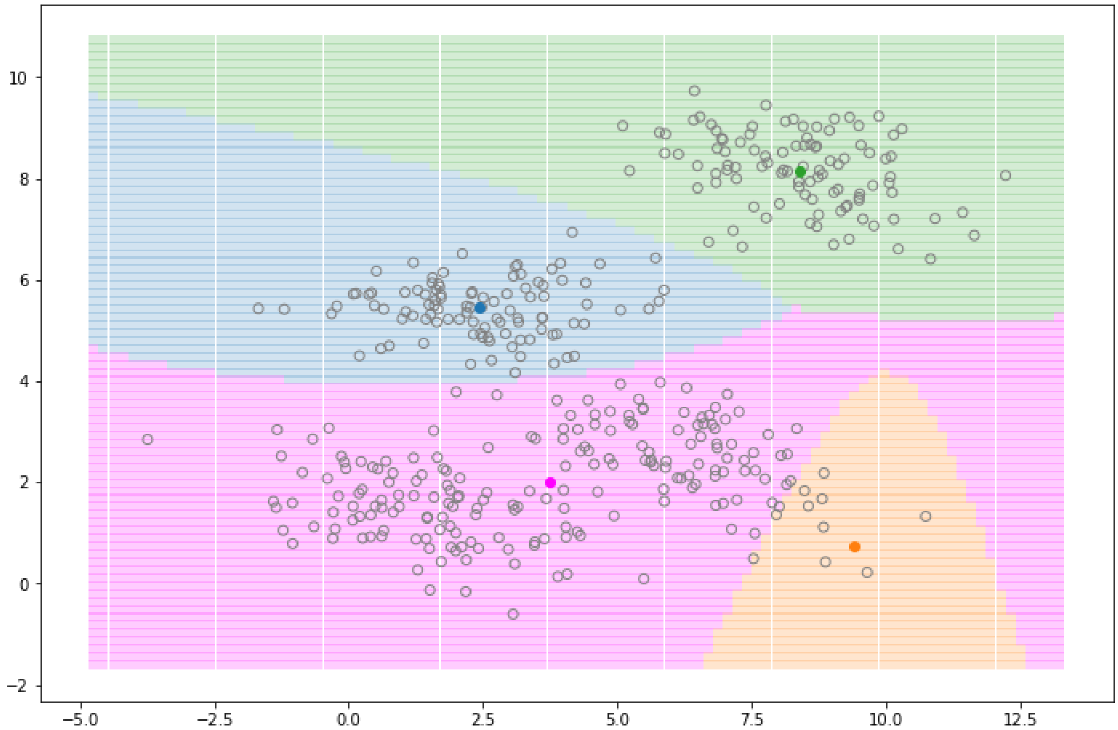

Iteration 13



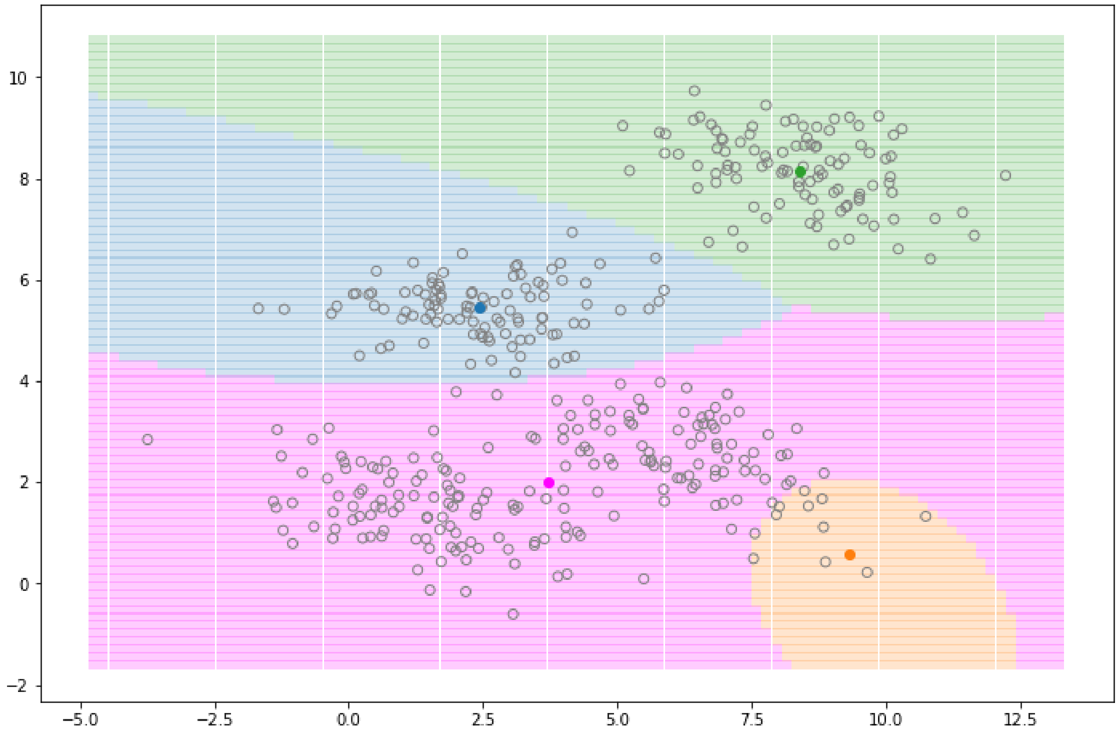

Iteration 14



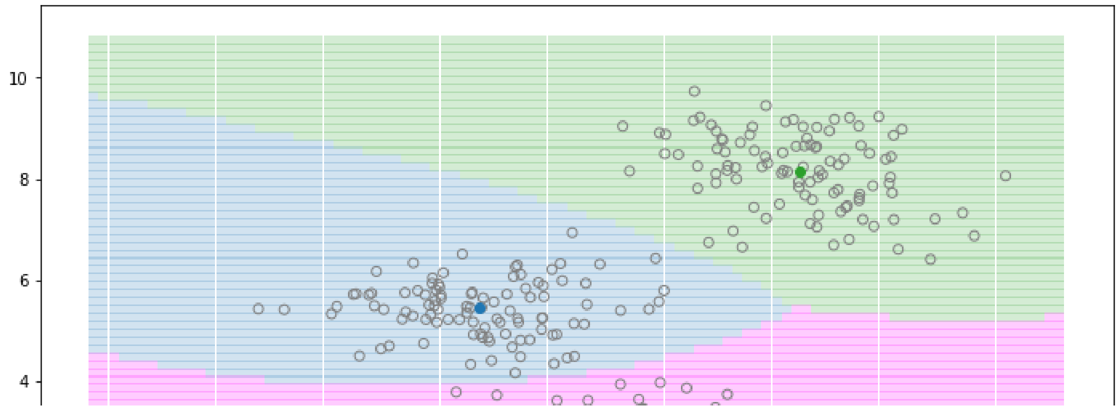
Iteration 15

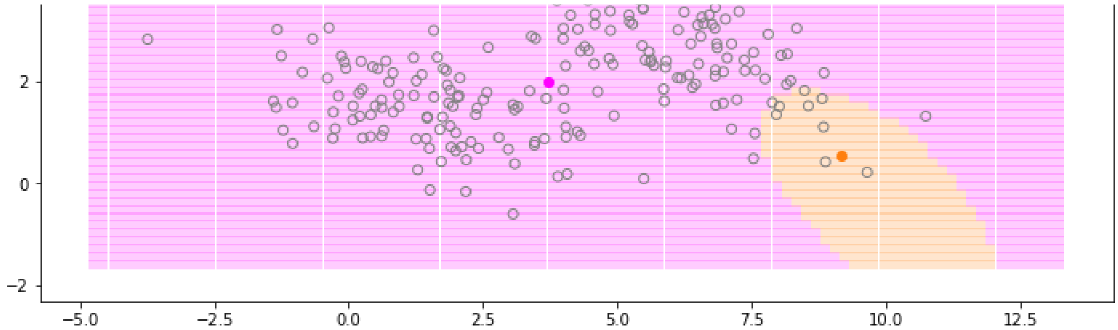



Iteration 16

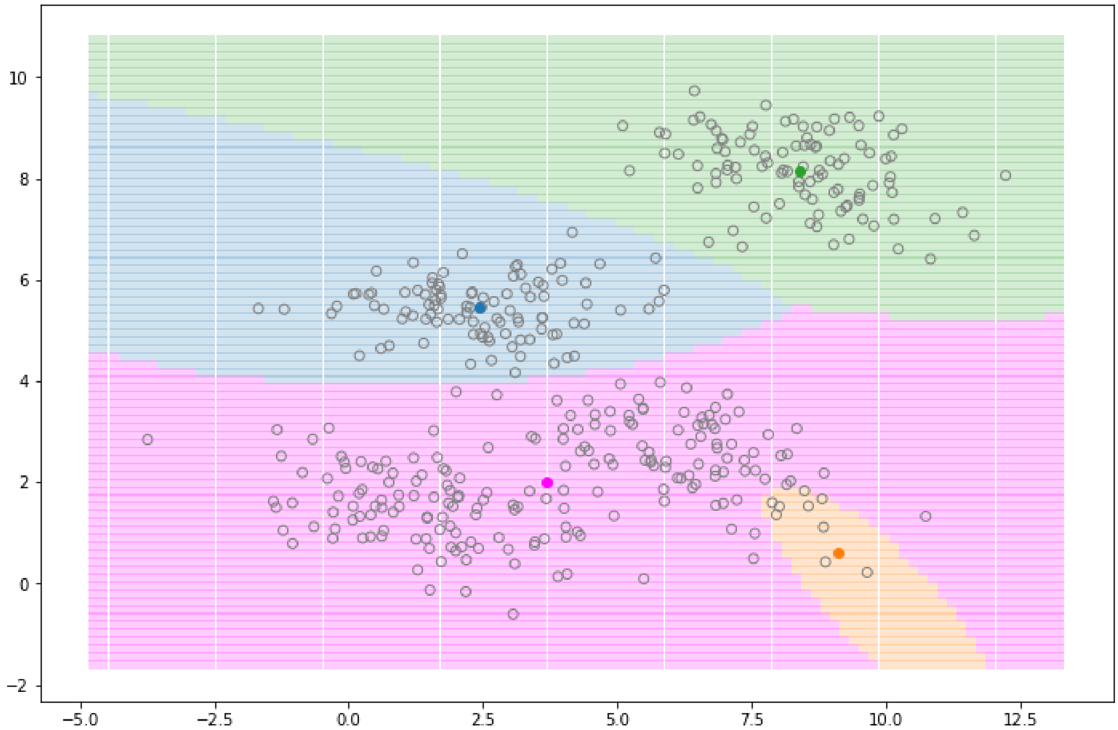



Iteration 17

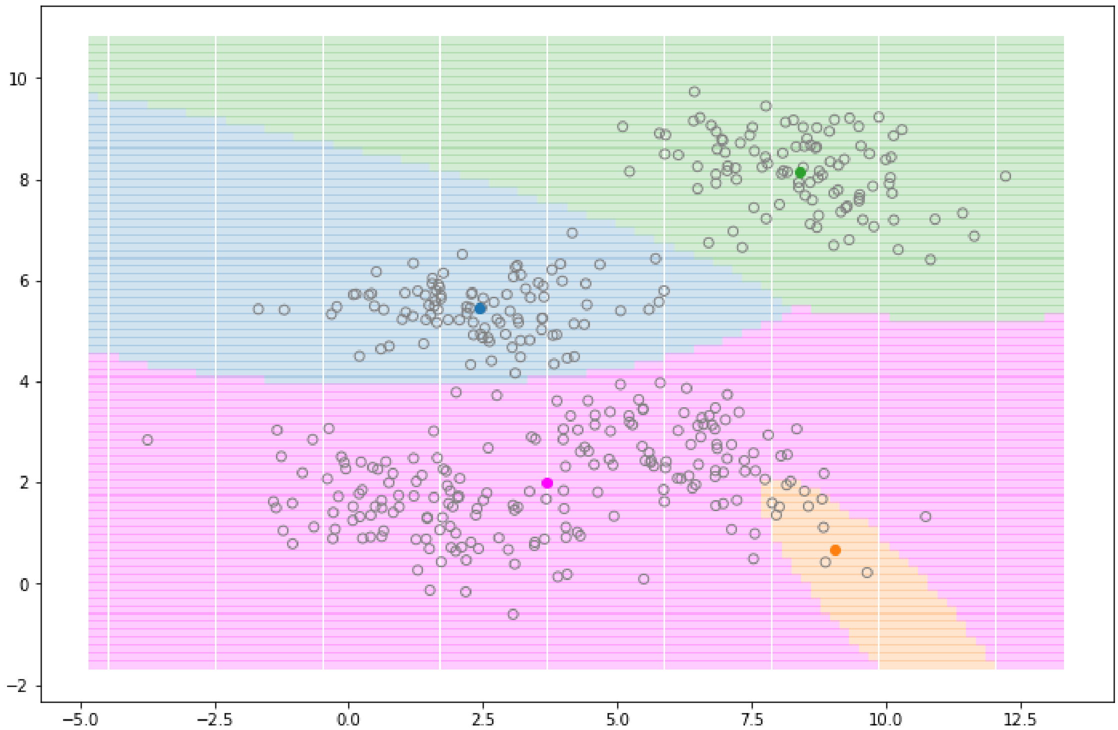




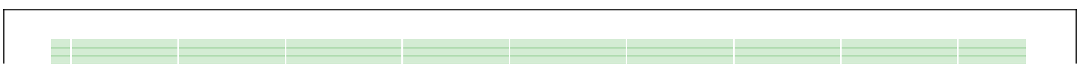

Iteration 18

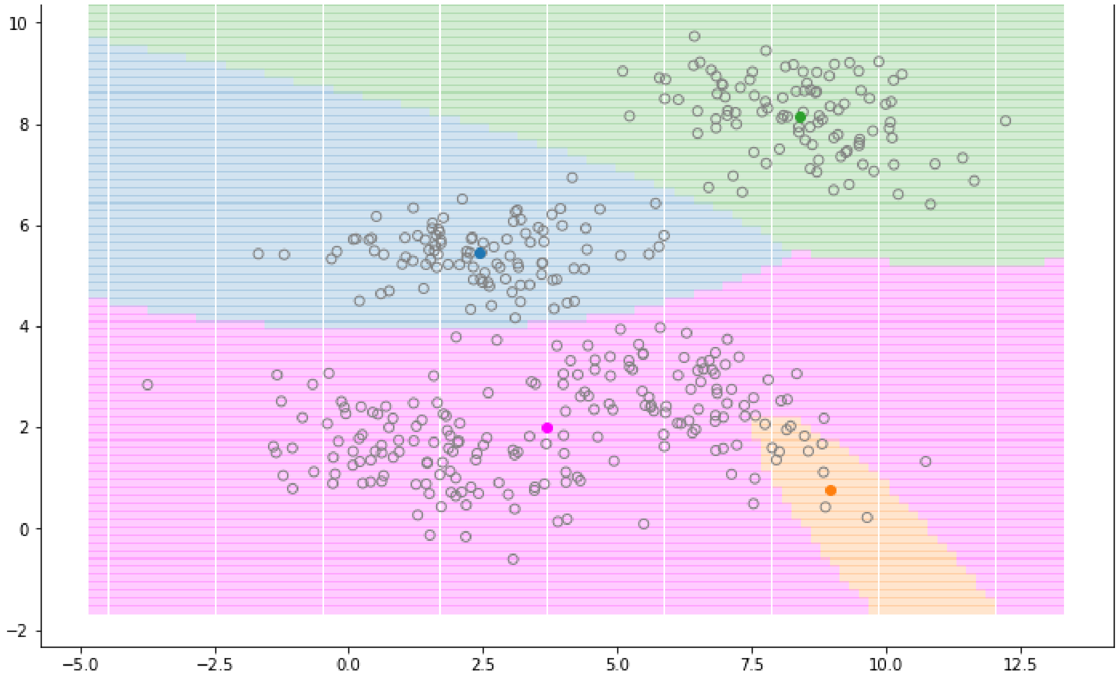



Iteration 19

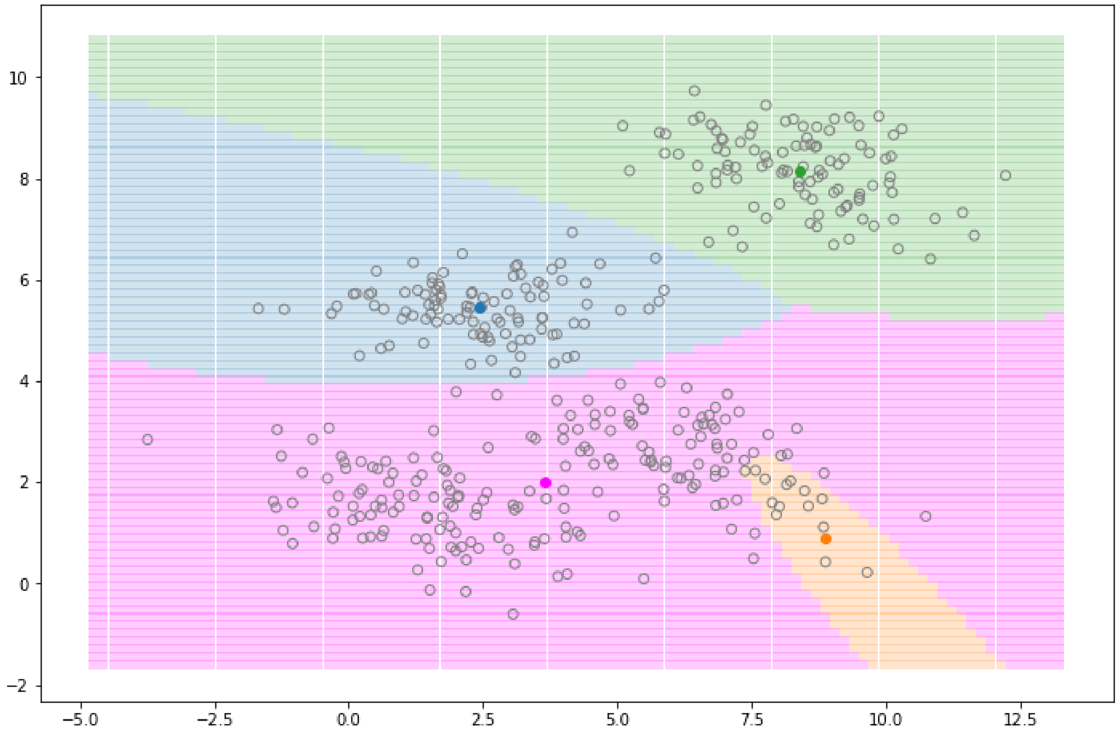



Iteration 20

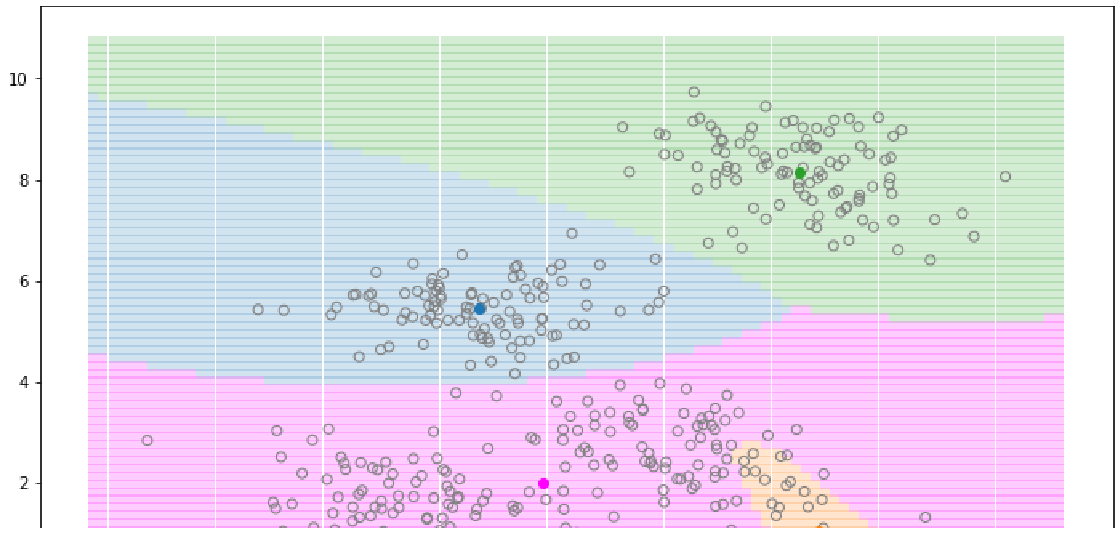


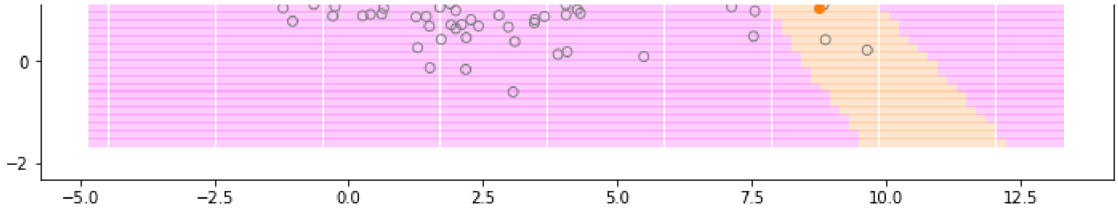



Iteration 21

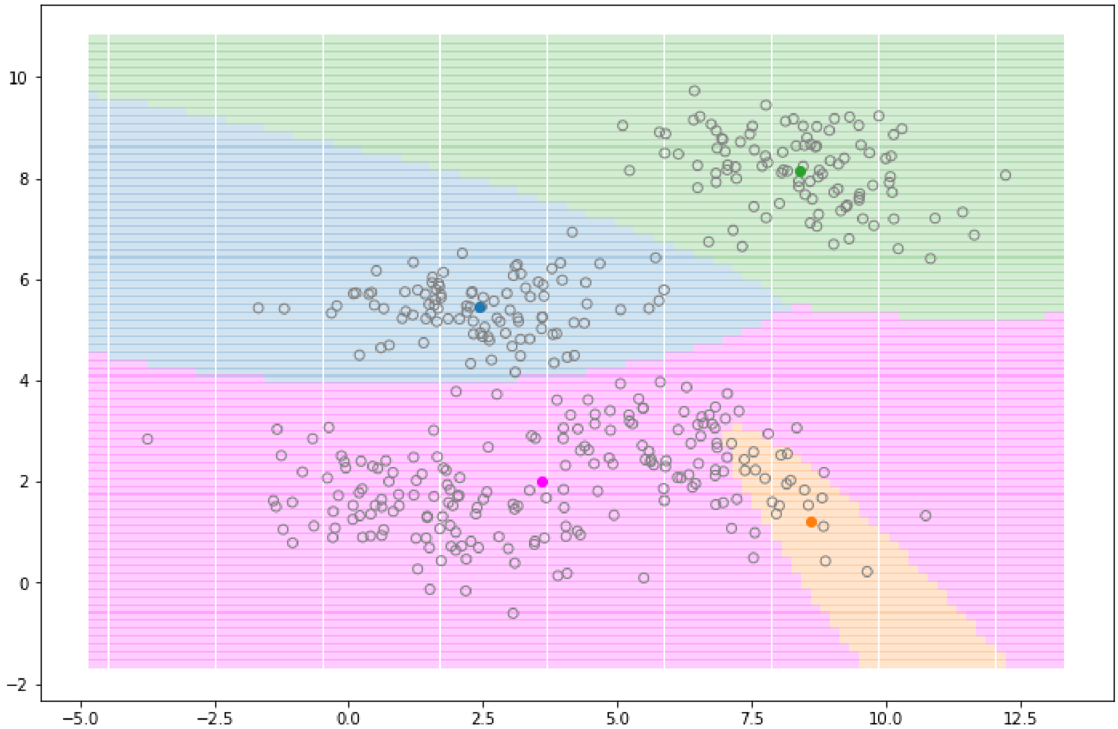



Iteration 22

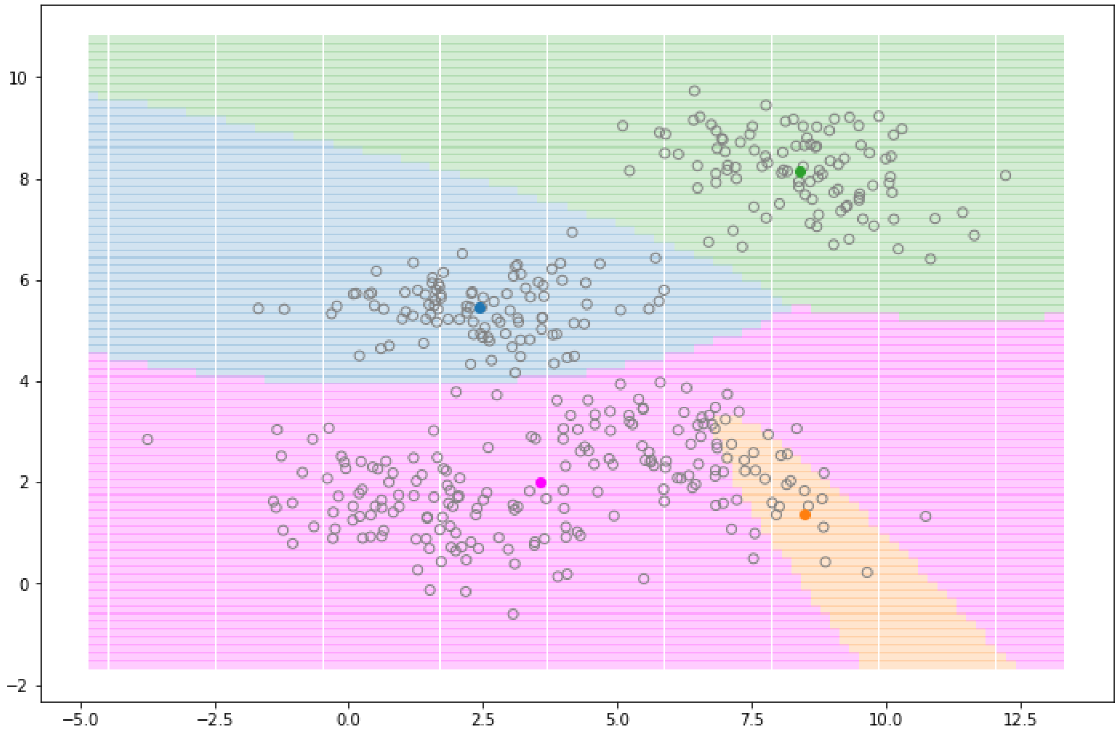




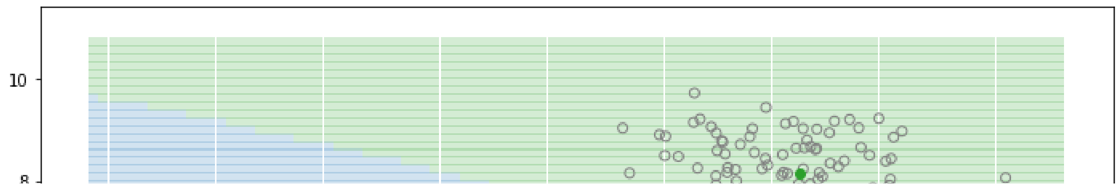

Iteration 23

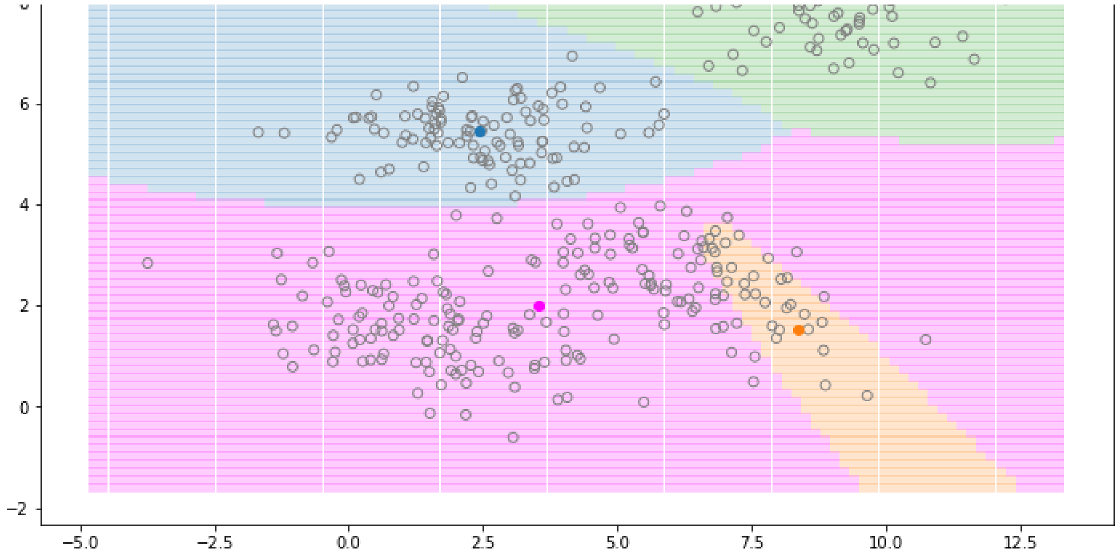



Iteration 24

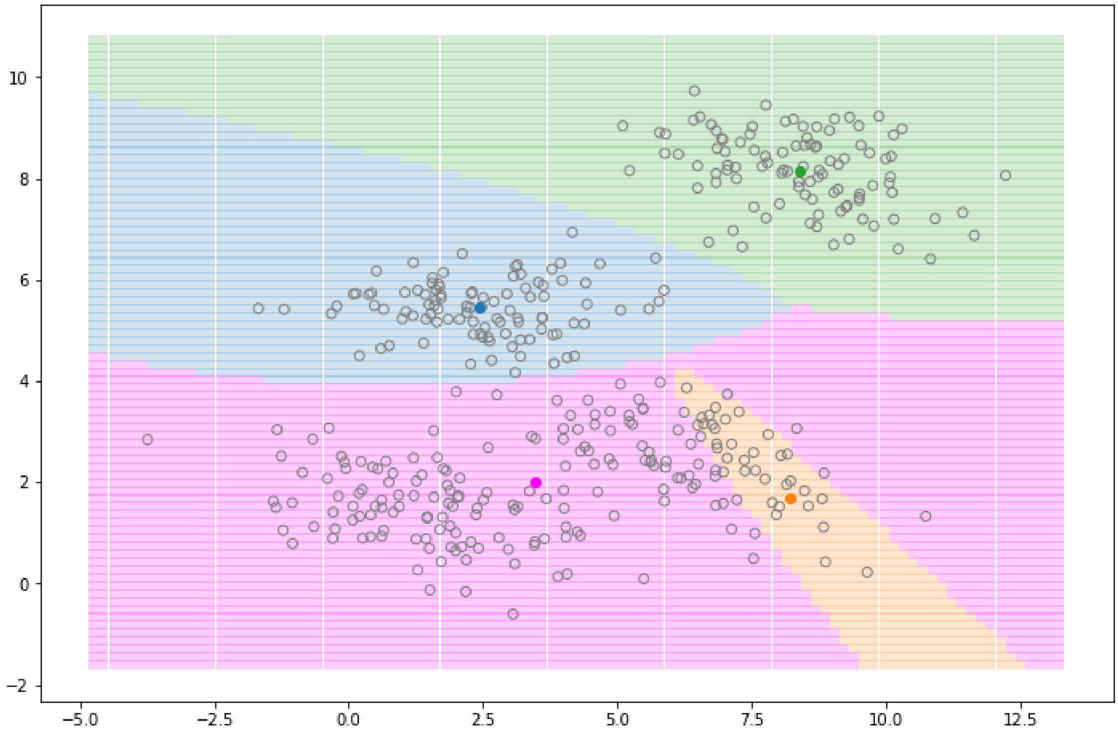



Iteration 25

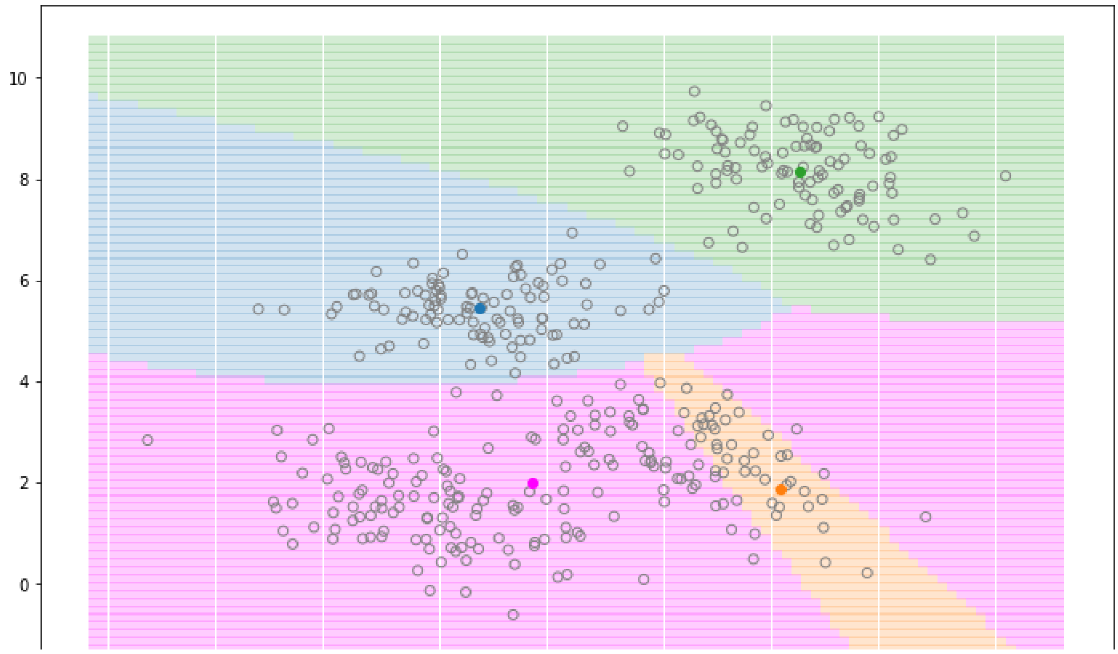




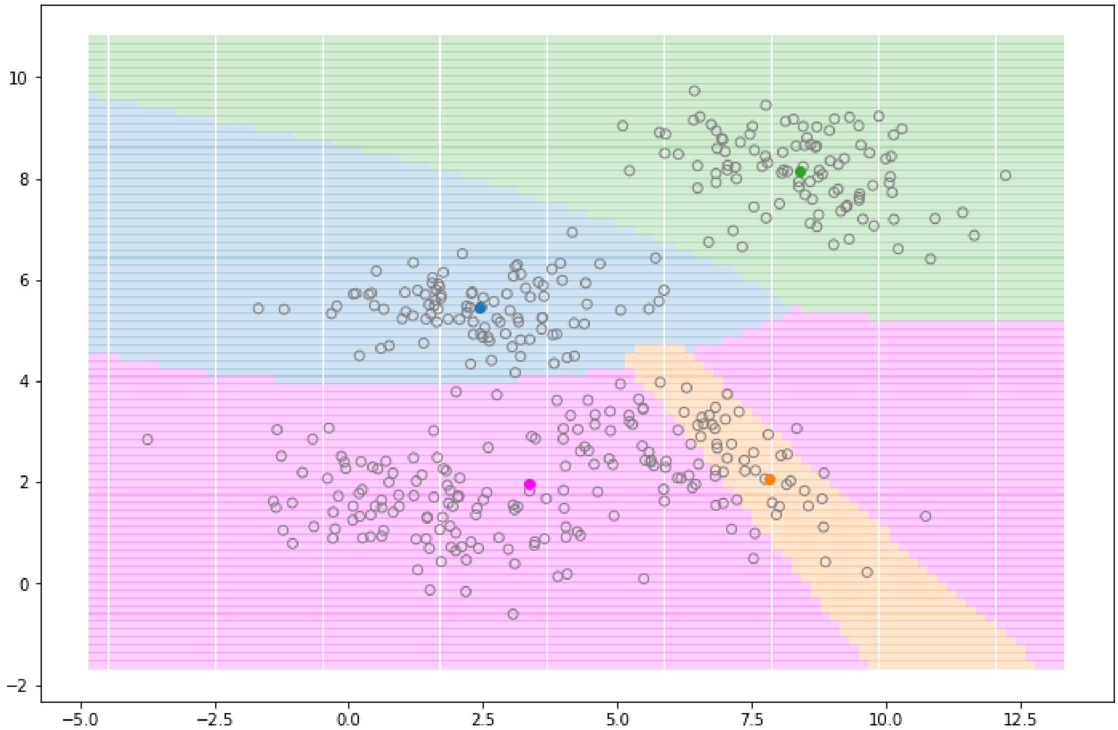

Iteration 26



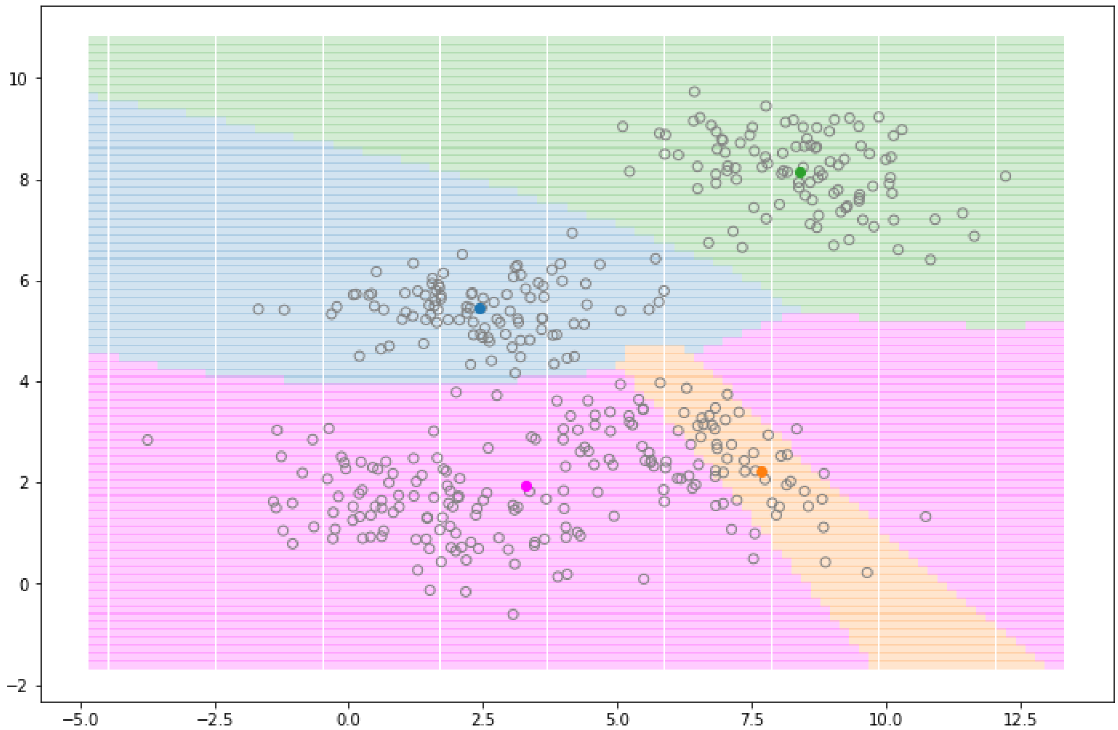

Iteration 27



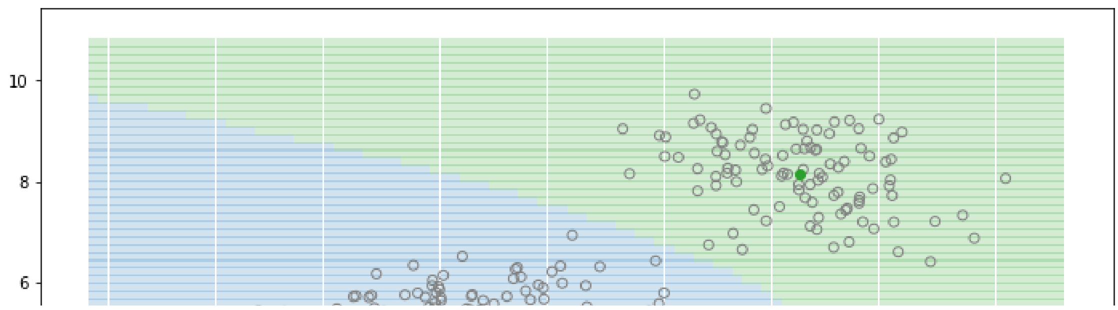

Iteration 28

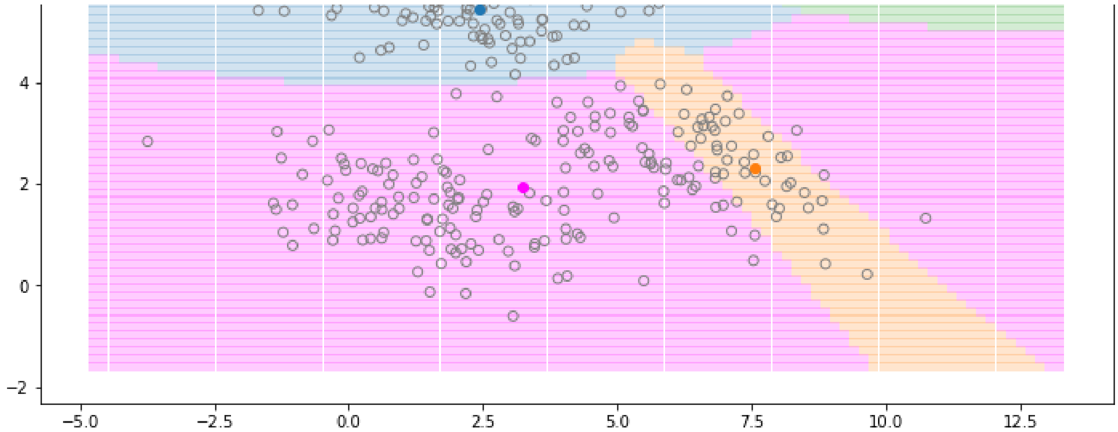



Iteration 29

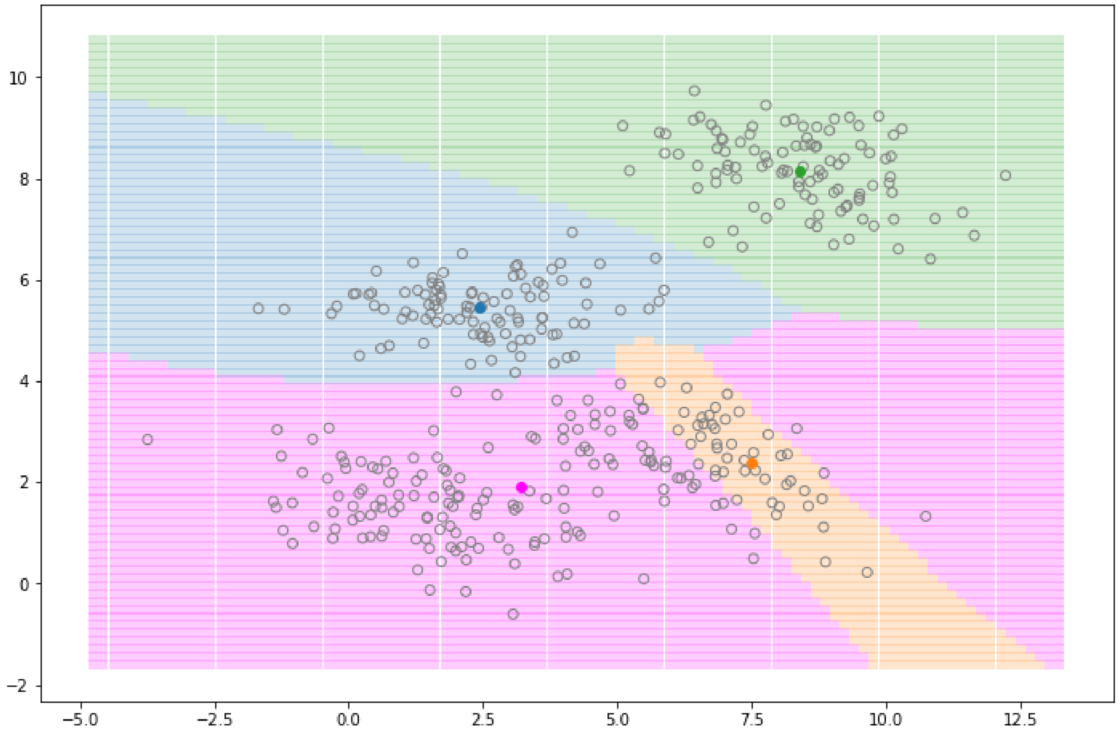



Iteration 30

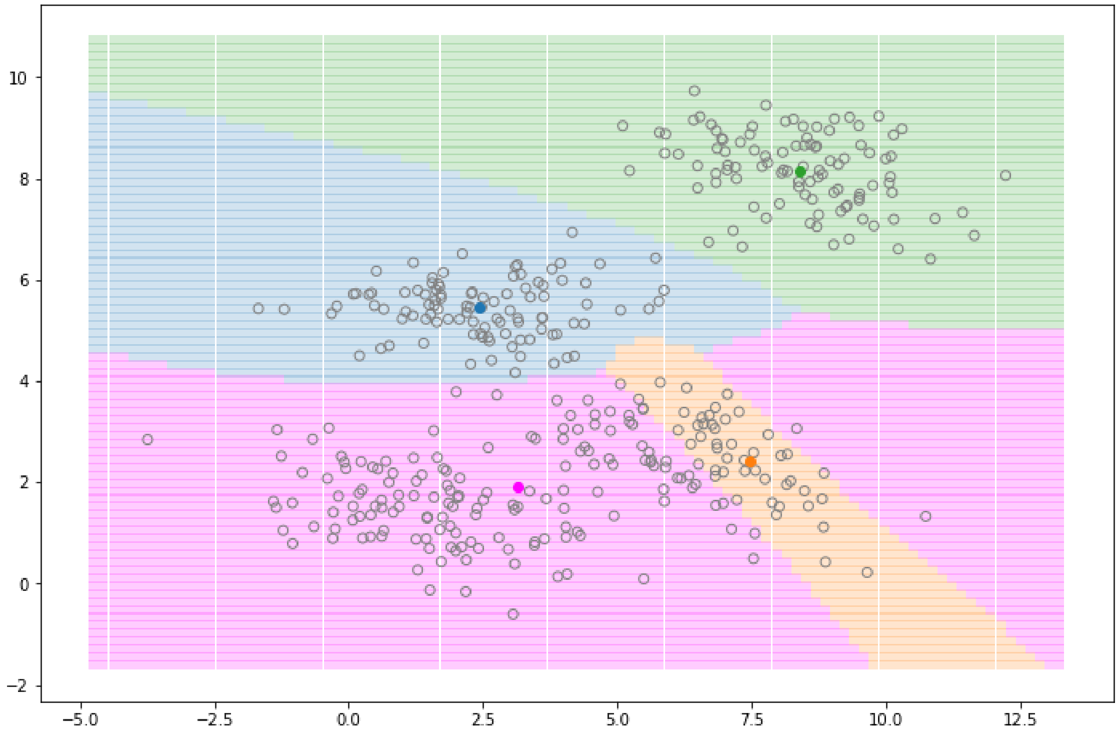




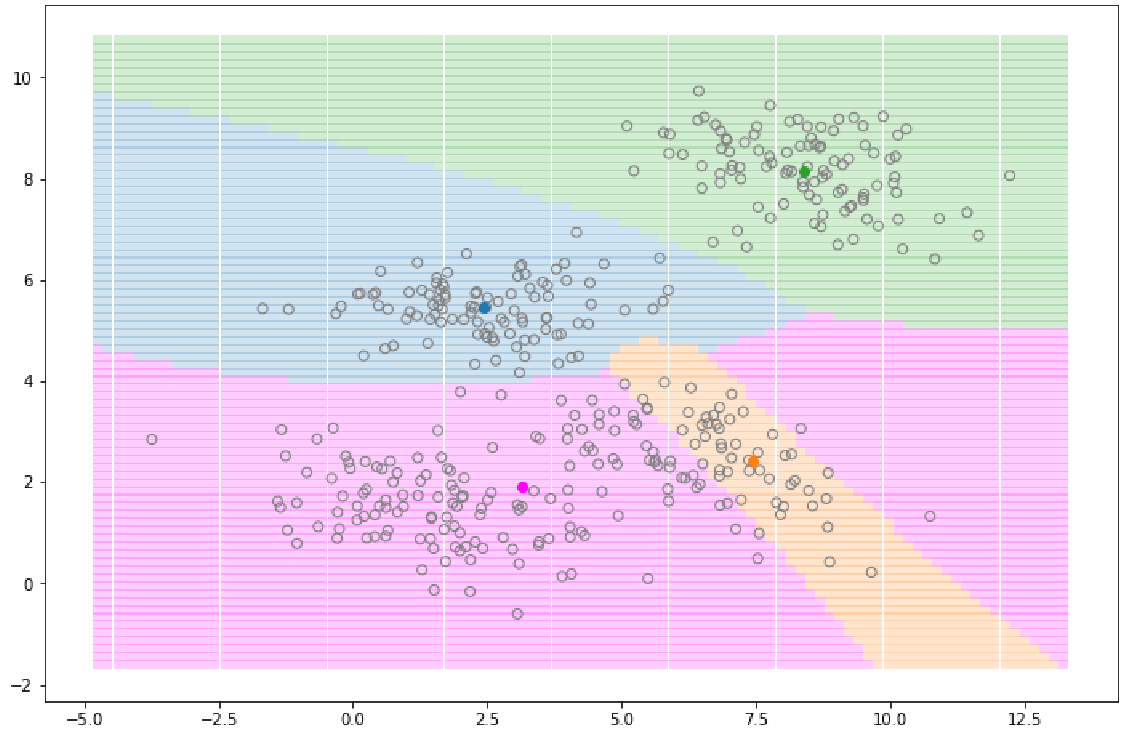

Iteration 31



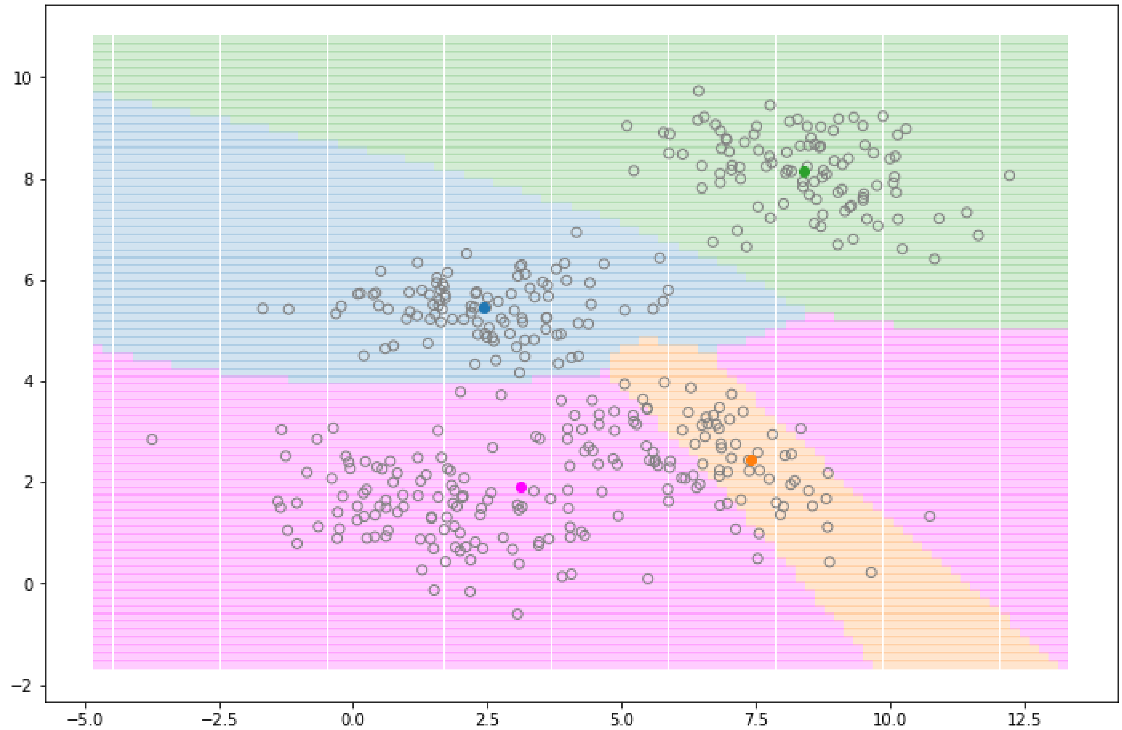

Iteration 32



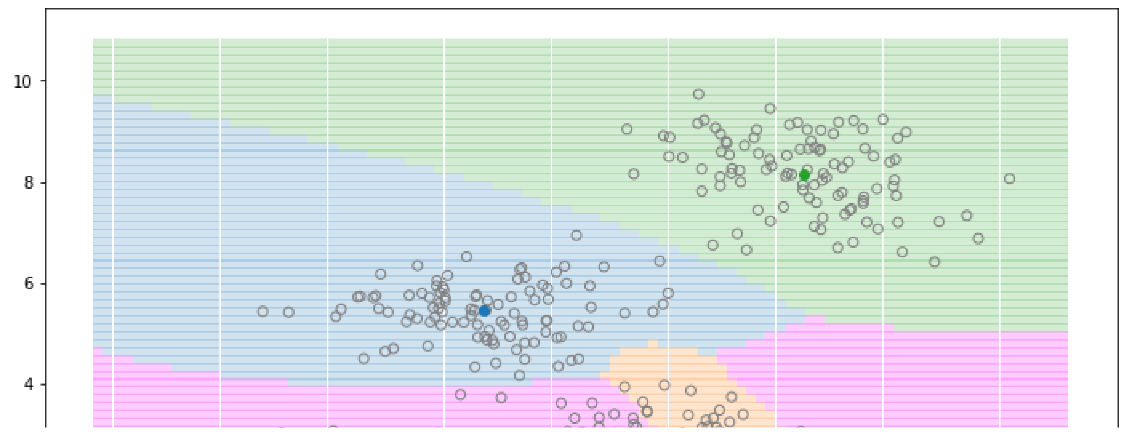
Iteration 33

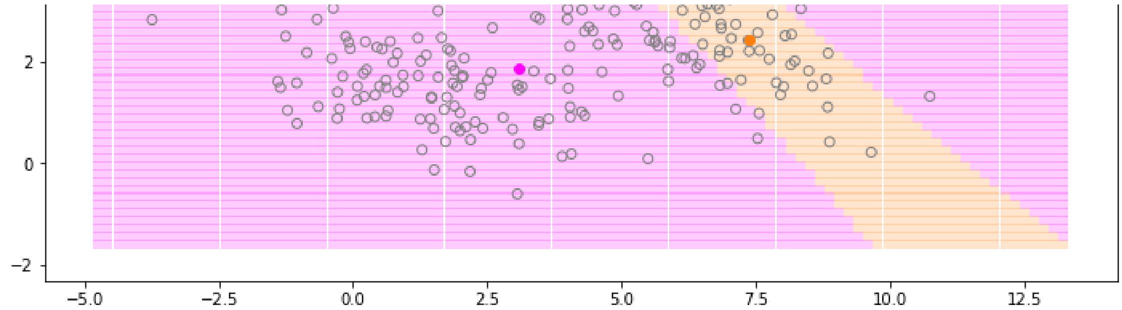



Iteration 34

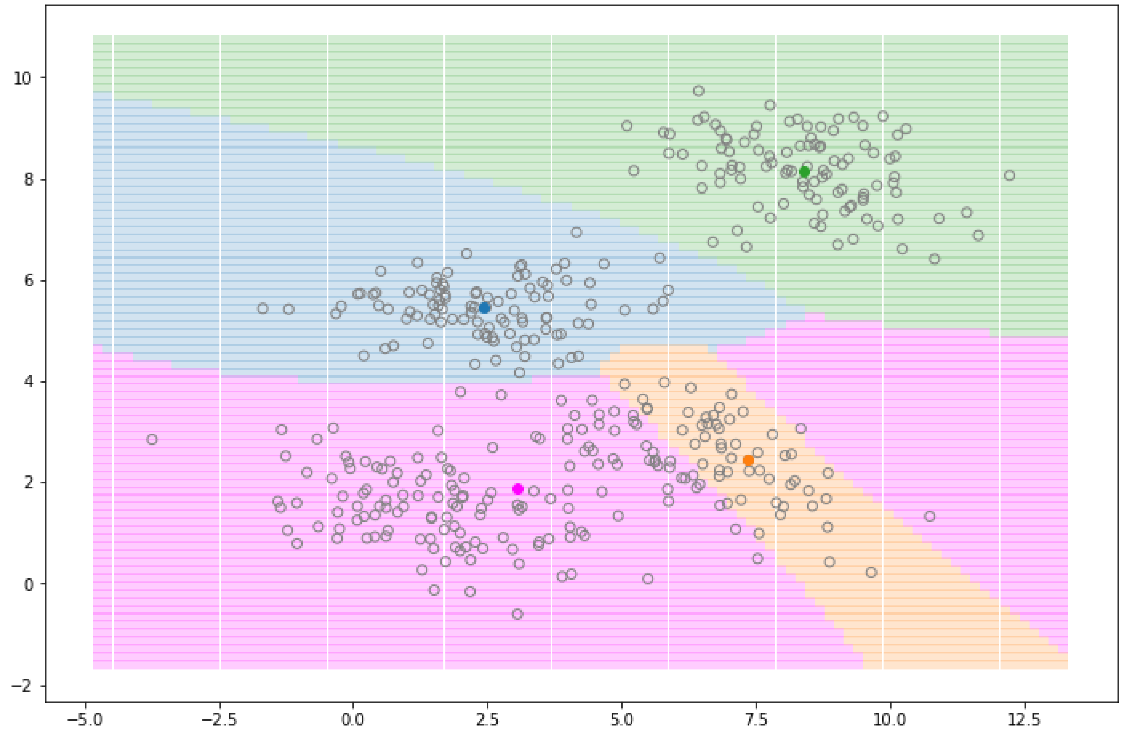



Iteration 35

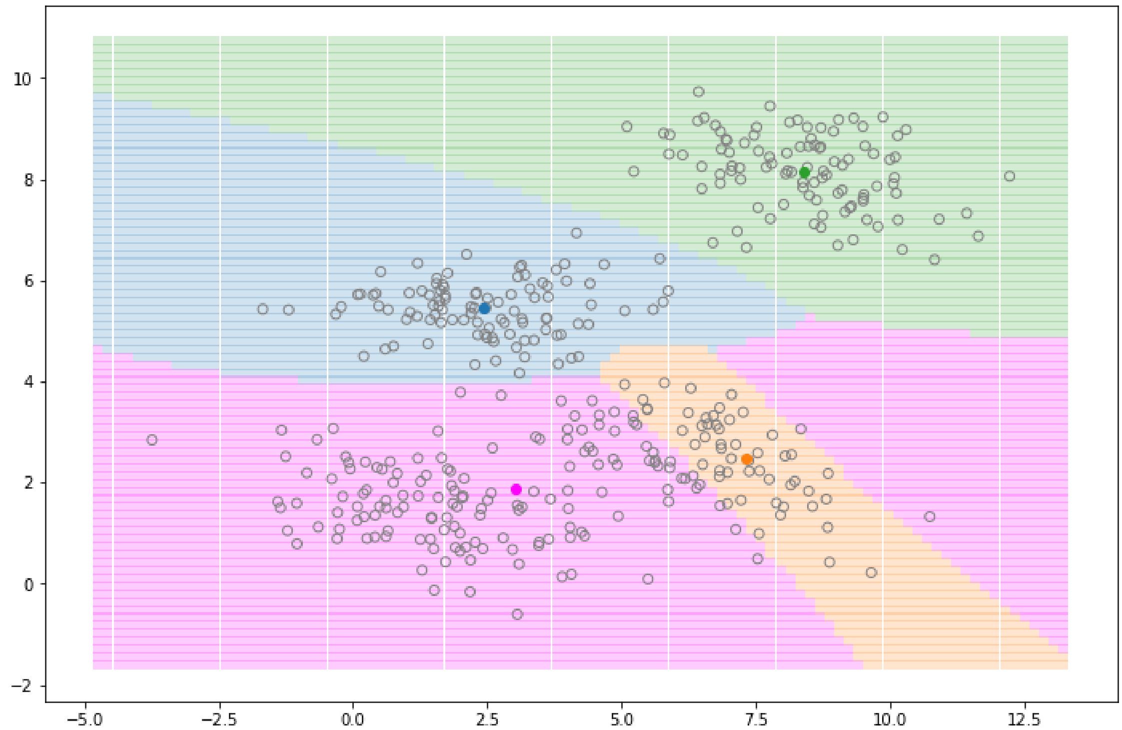




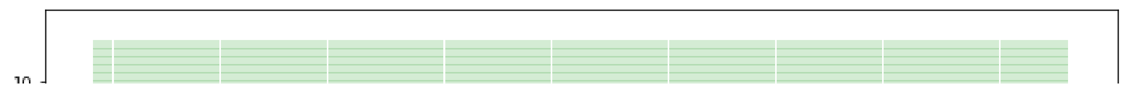

Iteration 36

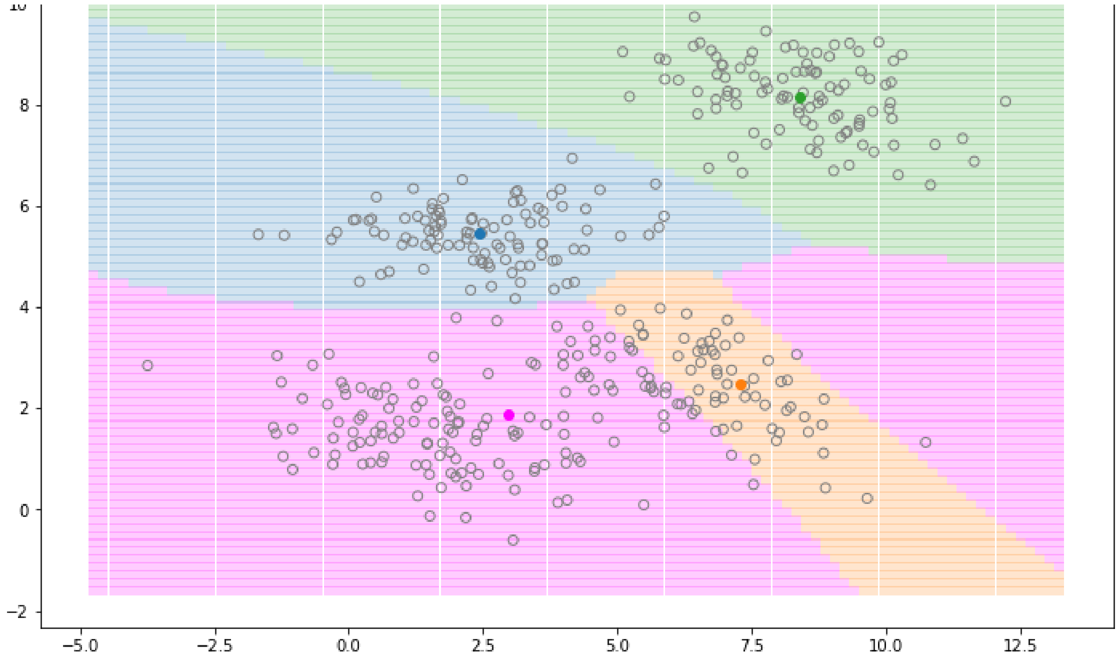


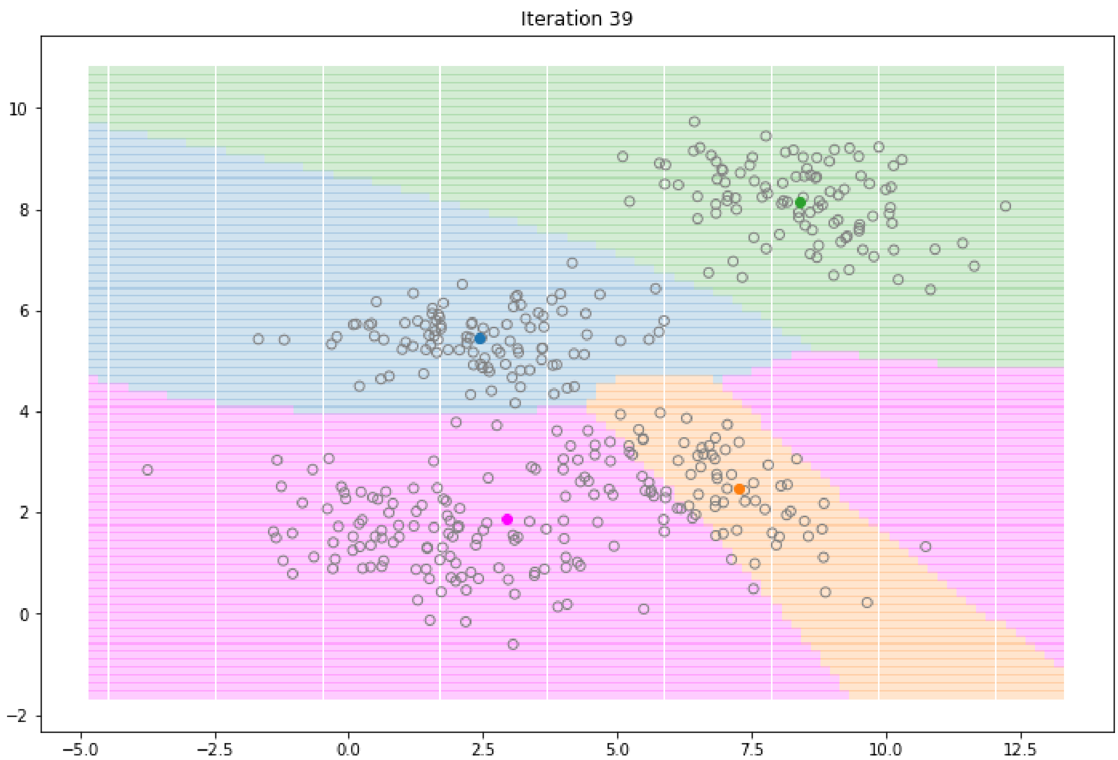

Iteration 37




Iteration 38






#### the 40 iterations plots for determining the clusters we generated

#### since in the assignment it was given to calcuate for only 2 gaussians to generate mixture model and learn (mu,pi,and covariance)

#### I generated 4 gaussians and learned the mixture model by learning (4 means,4 covariance,4 pi's) that generated 4 clusters

#### and the above 40 figures shows that how the clusters are learned as covariance, means, and pi's are updated at each likelihood step

In [ ]:

```

```

# IME692_Assignment_2

November 2, 2019

```
In [76]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns
         import sklearn
         import sys
         import gc
         from scipy.spatial import Voronoi
```

```
In [6]: np.random.seed(1)
```

```
In [5]: A = np.random.multivariate_normal([0,1],[[0.5,0],[0,0.5]],50)
```

```
In [15]: #checking the data
         A
```

```
Out[15]: array([[ 1.14858562,  0.56742289],
                [-0.37347383,  0.24129661],
                [ 0.6119356 , -0.62743362],
                [ 1.23376823,  0.46174544],
                [ 0.22559471,  0.82366852],
                [ 1.03386644, -0.45673947],
                [-0.22798339,  0.72843256],
                [ 0.80169606,  0.22225943],
                [-0.12192515,  0.37926036],
                [ 0.02984963,  1.41211259],
                [-0.77825528,  1.8094419 ],
                [ 0.63752091,  1.35531715],
                [ 0.63700135,  0.51653139],
                [-0.08689651,  0.33831109],
                [-0.18942548,  1.37501795],
                [-0.48907801,  0.71945289],
                [-0.48590448,  0.40234936],
                [-0.47464269,  0.99104478],
                [-0.79005772,  1.16575693],
                [ 1.17365737,  1.52470446],
                [-0.13564822,  0.37235154],
                [-0.5283207 ,  2.19674613],
```

```
       [ 0.03592651,  0.54957606],
       [ 0.13499763,  2.48510465],
       [ 0.08496521,  1.4364285 ],
       [ 0.21225247,  0.75092174],
       [-0.80788237,  0.75297739],
       [-0.14771053,  1.41480524],
       [ 0.59325086,  1.6583886 ],
       [ 0.20194073,  1.62588932],
       [-0.5334399 ,  1.88591157],
       [ 0.36269615,  0.78921653],
       [ 0.34543449,  0.94656273],
       [ 0.80018281,  2.07467278],
       [ 1.54543519,  0.01252797],
       [-1.02114266,  0.64328877],
       [ 0.1131633 ,  1.61954499],
       [ 0.22318761, -0.42991219],
       [-0.21651893,  1.58546648],
       [ 0.16270155,  1.53882327],
       [-0.15720974,  0.85804261],
       [ 0.13191882,  1.2899503 ],
       [ 0.14021908,  1.08415182],
       [-0.47422985,  1.26697791],
       [ 0.08614065,  1.79866573],
       [ 0.84776296,  1.13092536],
       [-0.26536653,  0.5483494 ],
       [ 0.29945573,  1.05468769],
       [-0.24314127,  1.03082763],
       [-0.4384068 ,  1.49358318]])
```

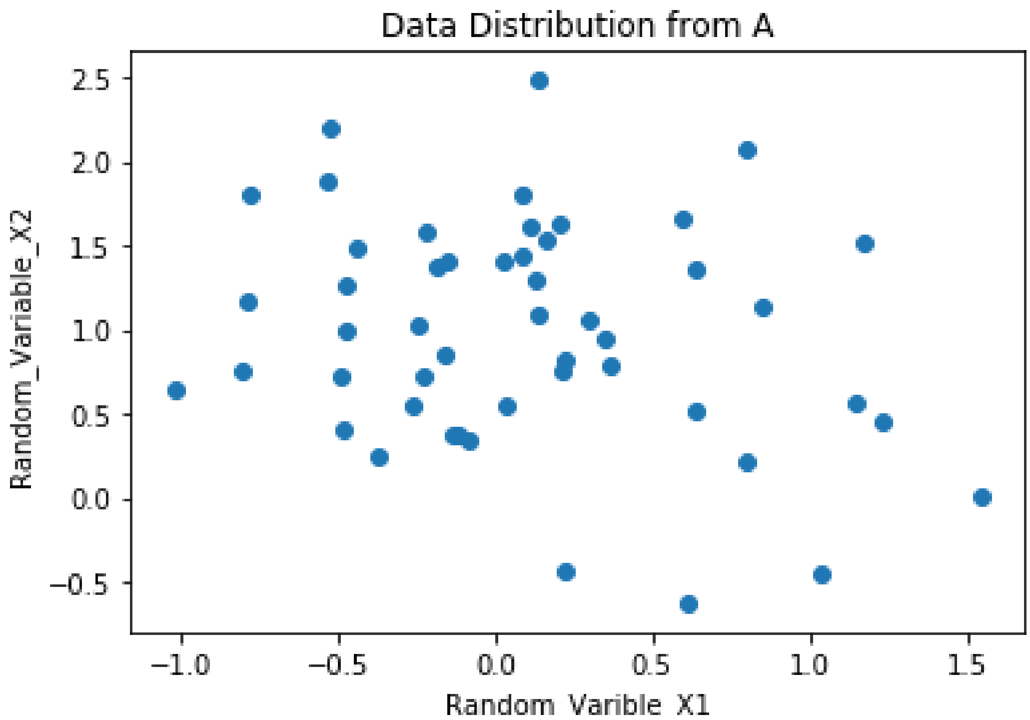
In [17]:
```
A[:,1]
```

Out[17]:
```
array([ 0.56742289,  0.24129661, -0.62743362,  0.46174544,  0.82366852,
       -0.45673947,  0.72843256,  0.22225943,  0.37926036,  1.41211259,
        1.8094419 ,  1.35531715,  0.51653139,  0.33831109,  1.37501795,
        0.71945289,  0.40234936,  0.99104478,  1.16575693,  1.52470446,
        0.37235154,  2.19674613,  0.54957606,  2.48510465,  1.4364285 ,
        0.75092174,  0.75297739,  1.41480524,  1.6583886 ,  1.62588932,
        1.88591157,  0.78921653,  0.94656273,  2.07467278,  0.01252797,
        0.64328877,  1.61954499, -0.42991219,  1.58546648,  1.53882327,
        0.85804261,  1.2899503 ,  1.08415182,  1.26697791,  1.79866573,
        1.13092536,  0.5483494 ,  1.05468769,  1.03082763,  1.49358318])
```

In [7]:
```
B = np.random.multivariate_normal([1,0],[[0.5,0],[0,0.5]],50)
```

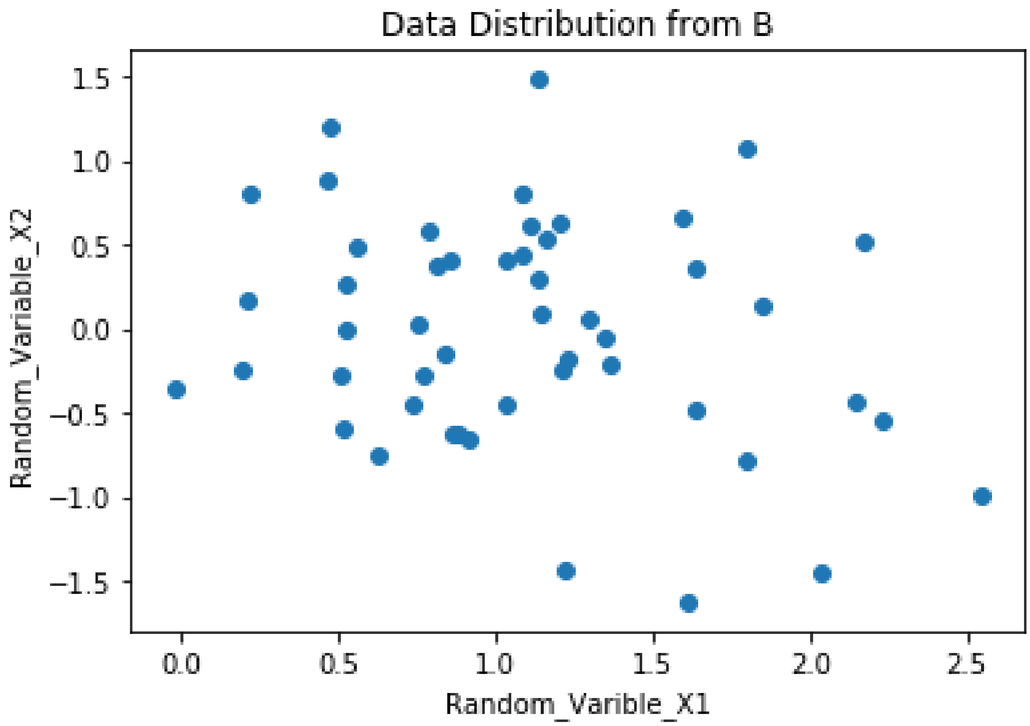
In [21]:
```
plt.scatter(A[:,0],A[:,1])
plt.xlabel('Random_Varible_X1')
plt.ylabel('Random_Variable_X2')
plt.title('Data Distribution from A')
```

Out[21]: Text(0.5, 1.0, 'Data Distribution from A')



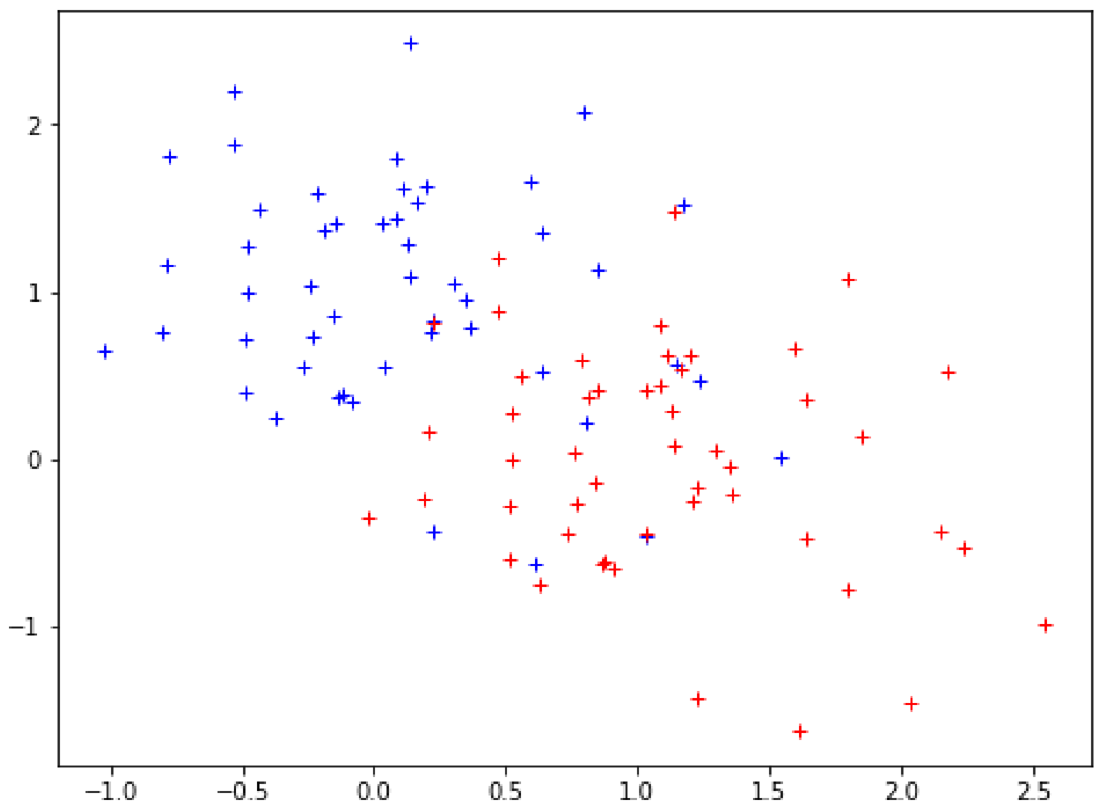

```
In [22]: plt.scatter(B[:,0],B[:,1])
         plt.xlabel('Random_Varible_X1')
         plt.ylabel('Random_Variable_X2')
         plt.title('Data Distribution from B')
```

Out[22]: Text(0.5, 1.0, 'Data Distribution from B')



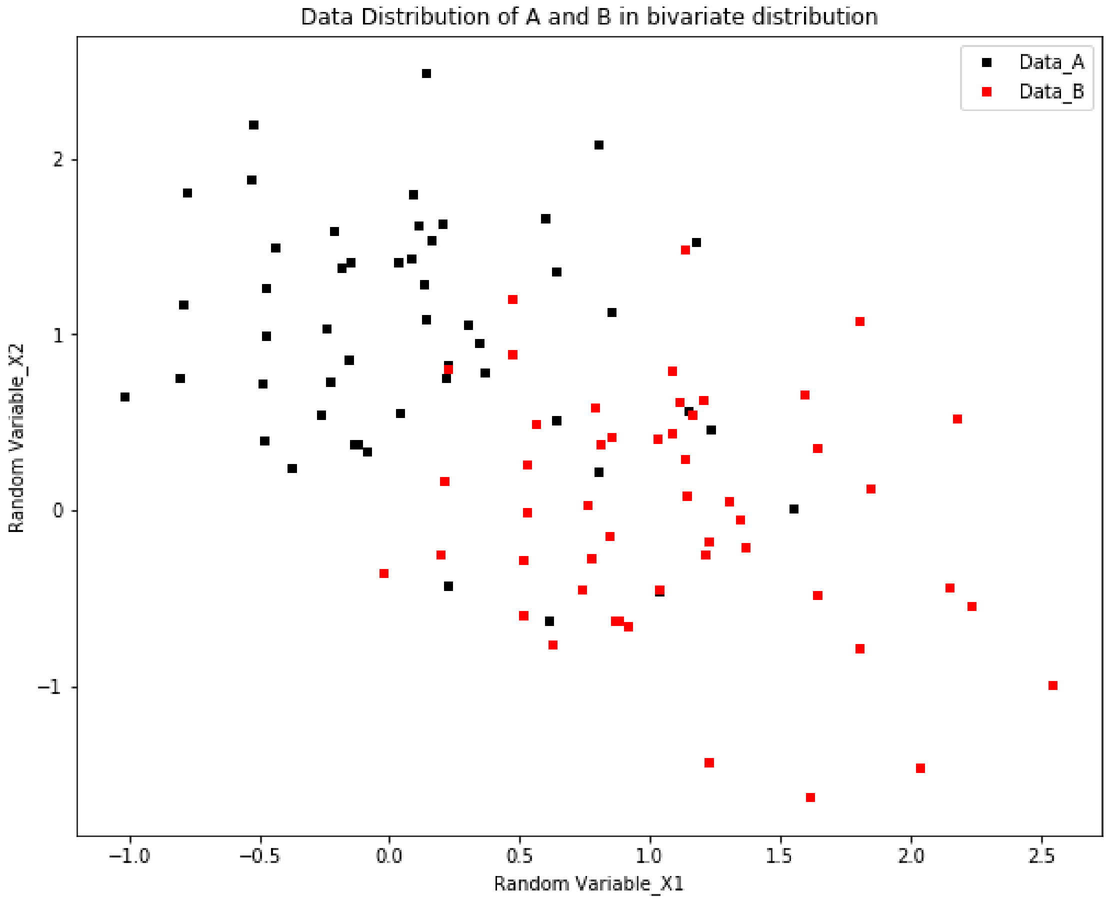

In [51]:
```
#Now plotting in the same plot (both of these plots)
#simple plot version
plt.figure(figsize = (8,6))
plt.plot(A[:,0],A[:,1],"b+",B[:,0],B[:,1],"r+")
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x5be0ed0>,
 <matplotlib.lines.Line2D at 0x5be0f90>]
```

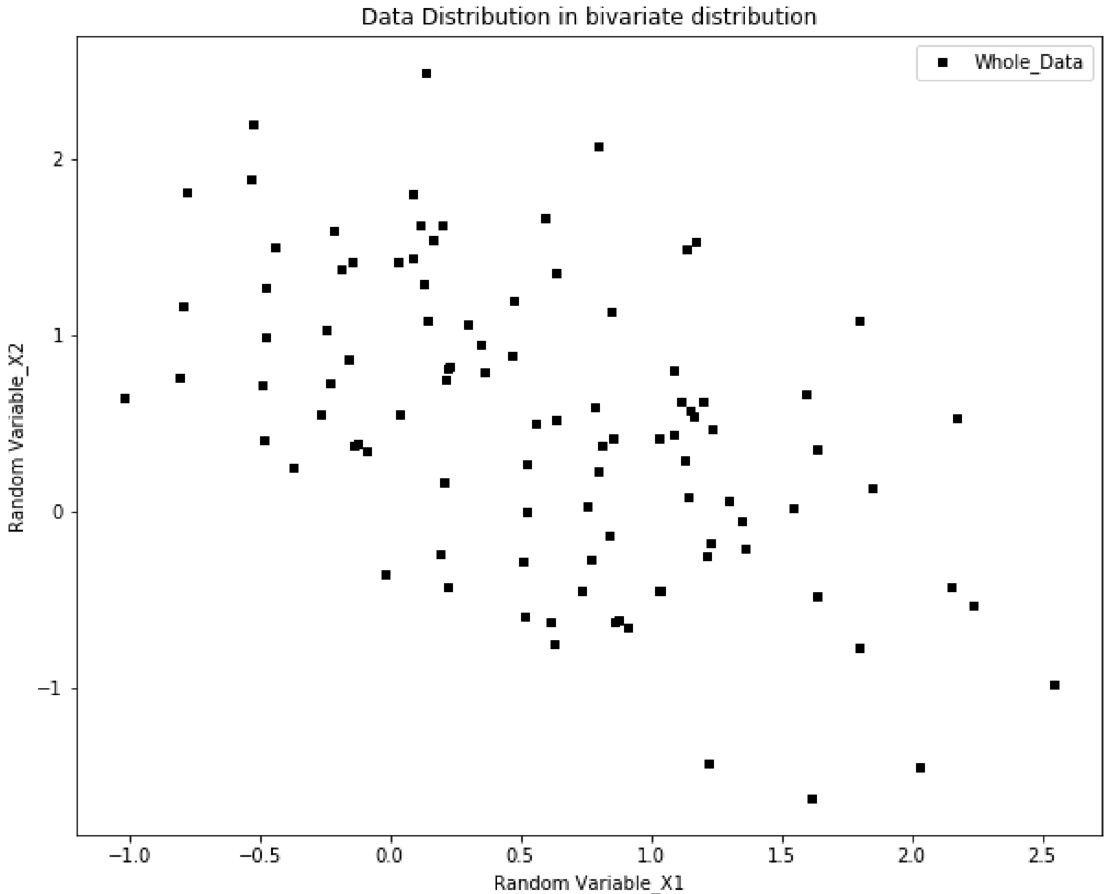


```
In [47]: #scatter version
         plt.figure(figsize  = (20,8))
         fig = plt.figure(figsize= (10,8))
         ax1 = fig.add_subplot(111)
         ax1.scatter(A[:,0],A[:,1],s=  10,c = 'black',marker = "s",label= "Data_A")
         ax1.scatter(B[:,0],B[:,1],s=10,c = 'r',marker = "s",label = "Data_B")
         plt.legend(loc = "upper right")
         plt.xlabel('Random Variable_X1')
         plt.ylabel('Random Variable_X2')
         plt.title("Data Distribution of A and B in bivariate distribution")
         plt.show()
```

```
<Figure size 1440x576 with 0 Axes>
```




```
In [52]: combined_data = np.concatenate((A,B))

In [54]: combined_data.shape

Out[54]: (100, 2)

In [77]: #scatter version
         # before training our data look like in this way
         plt.figure(figsize  = (20,8))
         fig = plt.figure(figsize= (10,8))
         ax1 = fig.add_subplot(111)
         ax1.scatter(combined_data[:,0],combined_data[:,1],s=  10,c = 'black',marker = "s",labe
         #ax1.scatter(B[:,0],B[:,1],s=10,c = 'black',marker = "s",label = "Data_B")
         plt.legend(loc = "upper right")
         plt.xlabel('Random Variable_X1')
         plt.ylabel('Random Variable_X2')
         plt.title("Data Distribution in bivariate distribution")
         plt.show()

<Figure size 1440x576 with 0 Axes>
```



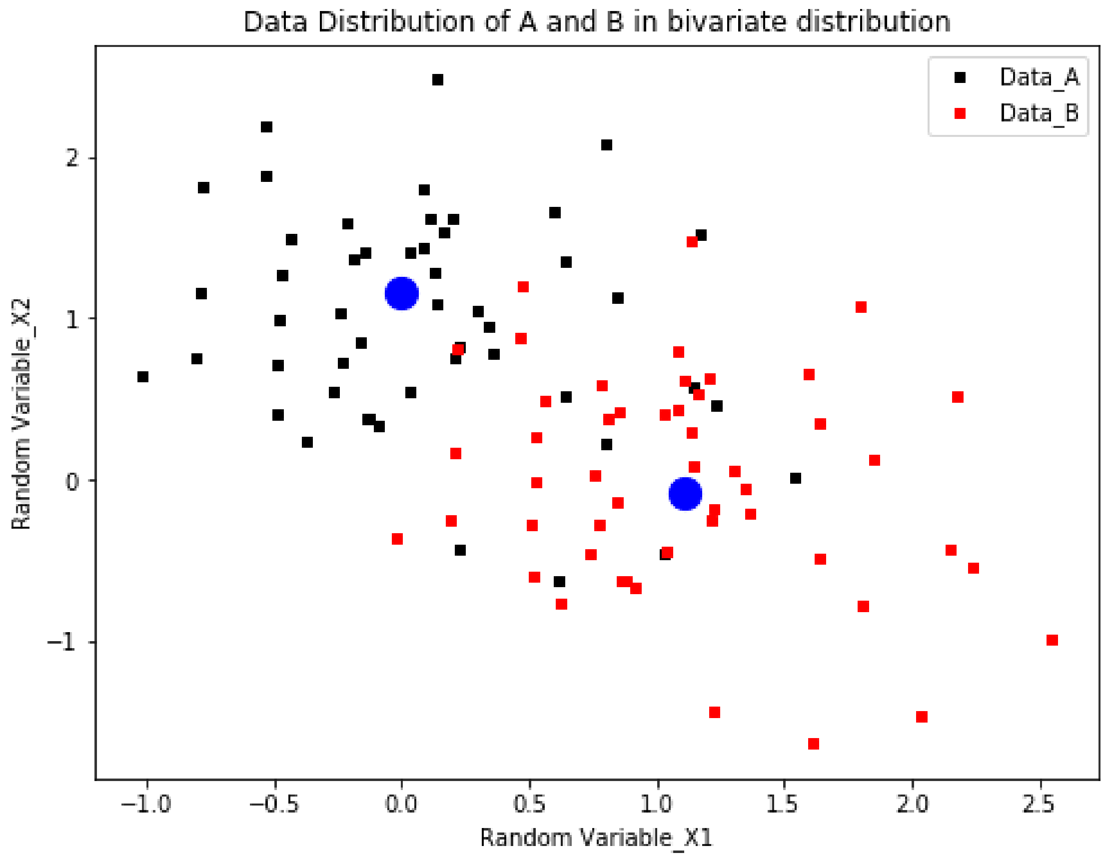

```
In [57]: # doing a random shuffling in the combined data for more
         #randomness and more abstract
         np.random.shuffle(combined_data)

In [ ]: combined_data

In [59]: from sklearn.cluster import KMeans

In [60]: kmeans = KMeans(n_clusters = 2, random_state = 0)

In [61]: kmeans.fit(combined_data)

Out[61]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
             n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
             random_state=0, tol=0.0001, verbose=0)

In [62]: k_means_labels = kmeans.labels_

In [69]: #showing the cluster centers in coordinates
         kmeans.cluster_centers_
```

```
Out[69]: array([[-0.00156623,  1.16361325],
                [ 1.107573  , -0.08695177]])
```

```
In [71]: #so point (-0.00156623, 1.16361325) is the first cluster which is for our data of A a
         # other one is for the second cluster centroid
```

```
In [74]: #scatter version
         plt.figure(figsize  = (20,8))
         fig = plt.figure(figsize= (8,6))
         ax1 = fig.add_subplot(111)
         ax1.scatter(A[:,0],A[:,1],s=  10,c = 'black',marker = "s",label= "Data_A")
         ax1.scatter(B[:,0],B[:,1],s=10,c = 'r',marker = "s",label = "Data_B")
         ax1.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s = 200, c= 'blu
         plt.legend(loc = "upper right")
         plt.xlabel('Random Variable_X1')
         plt.ylabel('Random Variable_X2')
         plt.title("Data Distribution of A and B in bivariate distribution")
         plt.show()
```

```
<Figure size 1440x576 with 0 Axes>
```



### 0.0.1 The Blue points are the cluster centroids for A and B respectively

```
In [78]: y_predict_A = kmeans.predict(A)
         y_predict_B = kmeans.predict(B)
```

```
In [79]: from sklearn.metrics import accuracy_score
```

```
In [81]: # Remember our sklearn KMeans attached "0" for cluster for data A and
         # "1" for data B
```

```
In [89]: #therefore the no of errors in the data A form kmeans clustering is
         count_error_in_A = 0
         yo= [] # this contains the index of the wrongly classified data points in dataset A
         count = 0
         for w in y_predict_A:
             count = count+1
             if (w!=0):
                 yo.append(count)
                 count_error_in_A = count_error_in_A+1
```

```
In [84]: count_error_in_A
```

```
Out[84]: 8
```

```
In [85]: #so there are 8 errors in the first data form kmeans clustering algo for k = 2
```

```
In [111]: # similarly for data B
          count_error_in_B = 0
          countB  = -1
          error_B = []
          for w in y_predict_B:
              countB = countB+1
              if (w!=1):
                  error_B.append(countB)
                  count_error_in_B = count_error_in_B+1

          count_error_in_B
```

```
Out[111]: 4
```

```
In [87]: # so there are 4 errors in dataset B
```

```
In [88]: # therefore we have a total error of 12 combined form dataset A and Dataset B
```

```
In [97]: error_index_A = []
         for w in yo:
             w = w-1
             error_index_A.append(w)
```

```
In [91]: y_predict_A
```

```
Out[91]: array([1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0])
```

```
In [98]: y_predict_A[error_index_A]
```

```
Out[98]: array([1, 1, 1, 1, 1, 1, 1, 1])
```

```
In [99]: error_index_A
```

```
Out[99]: [0, 2, 3, 5, 7, 12, 34, 37]
```

```
In [101]: y_predict_A[49]
```

```
Out[101]: 0
```

```
In [104]: new_A = A.copy()
```

```
In [109]: new_A = np.delete(new_A,error_index_A,axis = 0)
```

```
In [110]: new_A.shape
```

```
Out[110]: (42, 2)
```

```
In [112]: new_B = B.copy()
```

```
In [113]: new_B = np.delete(new_B,error_B,axis = 0)
```

```
In [114]: new_B.shape
```

```
Out[114]: (46, 2)
```

```
In [117]: wrong_A = np.take(A,error_index_A,axis = 0)
```

```
In [118]: wrong_A
```

```
Out[118]: array([[ 1.14858562,  0.56742289],
                 [ 0.6119356 , -0.62743362],
                 [ 1.23376823,  0.46174544],
                 [ 1.03386644, -0.45673947],
                 [ 0.80169606,  0.22225943],
                 [ 0.63700135,  0.51653139],
                 [ 1.54543519,  0.01252797],
                 [ 0.22318761, -0.42991219]])
```

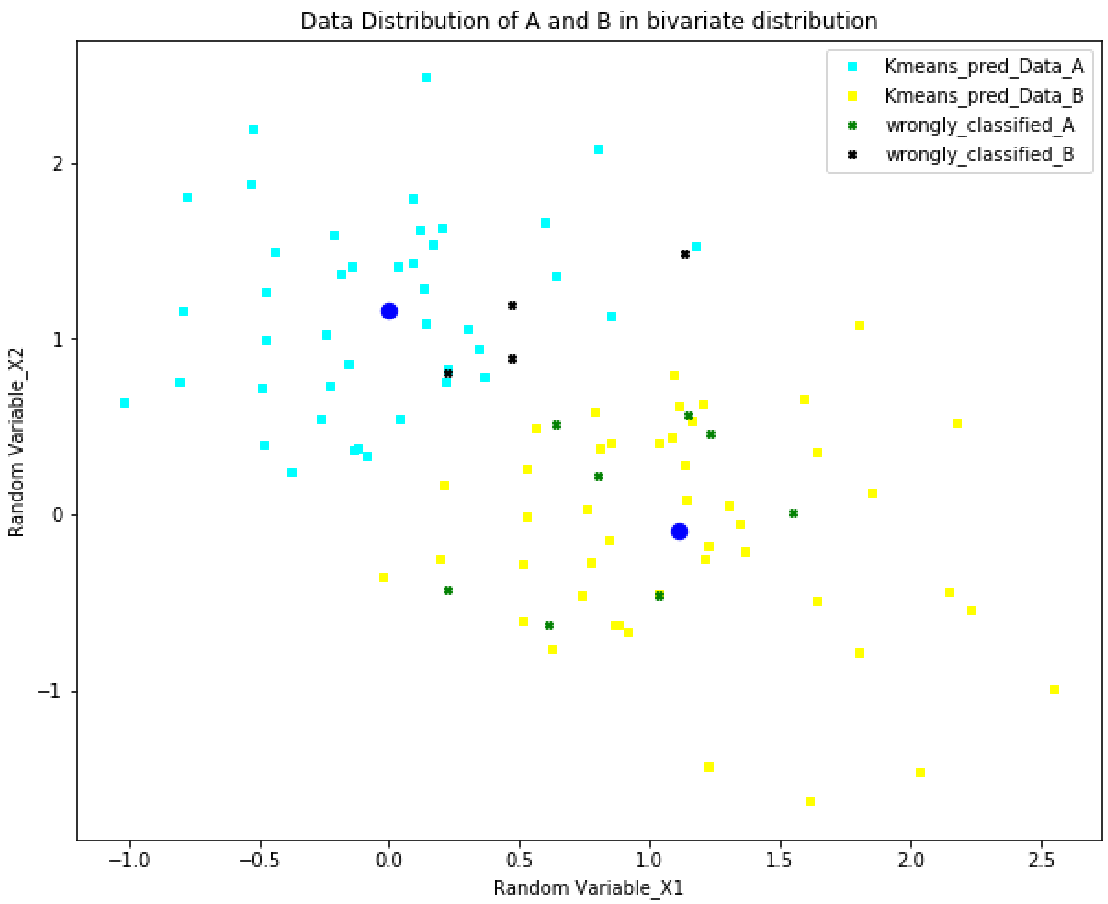
```
In [120]: wrong_B = np.take(B,error_B,axis = 0)
```

```
In [ ]:
```

```
In [115]: #now plotting the new_predicted data according to the KMeans algo
```

In [143]:
```
#scatter version
plt.figure(figsize  = (20,8))
fig = plt.figure(figsize= (10,8))
ax1 = fig.add_subplot(111)
ax1.scatter(new_A[:,0],new_A[:,1],s=  10,c = 'cyan',marker = "s",label= "Kmeans_pred_
ax1.scatter(new_B[:,0],new_B[:,1],s=10,c = 'yellow',marker = "s",label = "Kmeans_pred
ax1.scatter(wrong_A[:,0],wrong_A[:,1],s=16,c = 'green',marker = "X",label = "wrongly_
ax1.scatter(wrong_B[:,0],wrong_B[:,1],s=16,c = 'black',marker = "X",label = "wrongly_
ax1.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s = 70, c= 'blu
plt.legend(loc = "upper right")
plt.xlabel('Random Variable_X1')
plt.ylabel('Random Variable_X2')
plt.title("Data Distribution of A and B in bivariate distribution")
plt.show()
```

<Figure size 1440x576 with 0 Axes>




In [ ]: