Solution 1:

In [260]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from numpy import linalg as LA
```

In [261]:
```
img_1 = Image.open("ColorPlane.jpg")
img_2 = Image.open("ColorBird.jpg")

width_1 = img_1.width
height_1 = img_1.height

width_2 = img_2.width
height_2 = img_2.height
```

In [262]:
```
col = [[250, 250, 250], [0, 0, 0], [250, 0, 0], [0, 250, 0], [0, 0, 250]]
```

In [263]:
```
feature_1 = []
feature_2 = []

pixel_1 = []
pixel_2 = []

for i in range(height_1):
    pixel_1.append([])
    for j in range(width_1):
        d = img_1.getpixel((j, i))
        pixel_1[i].append([d[0], d[1], d[2]])

for i in range(height_2):
    pixel_2.append([])
    for j in range(width_2):
        d = img_2.getpixel((j, i))
        pixel_2[i].append([d[0], d[1], d[2]])

pix_1 = np.array(pixel_1)
pix_2 = np.array(pixel_2)

max_r_1 = np.max(pix_1[:, 0])
max_g_1 = np.max(pix_1[:, 1])
max_b_1 = np.max(pix_1[:, 2])

max_r_2 = np.max(pix_2[:, 0])
max_g_2 = np.max(pix_2[:, 1])
max_b_2 = np.max(pix_2[:, 2])

for i in range(height_1):
    for j in range(width_1):
        d = img_1.getpixel((j, i))
        feature_1.append([j/width_1, i/height_1, d[0]/max_r_1, d[1]/max_g_1, d
[2]/max_b_1])

for i in range(height_2):
    for j in range(width_2):
        d = img_2.getpixel((j, i))
        feature_2.append([j/width_2, i/height_2, d[0]/max_r_2, d[1]/max_g_2, d
[2]/max_b_2])
```

In [264]:
```
col[labels_1[width_1*i + j]]
```

Out[264]:
```
[0, 250, 0]
```

In [265]:
```
def k_mean_func(feature_1, feature_2, K):
    kmeans_1 = KMeans(n_clusters = K).fit(np.array(feature_1))
    labels_1 = kmeans_1.labels_

    kmeans_2 = KMeans(n_clusters = K).fit(np.array(feature_2))
    labels_2 = kmeans_2.labels_

    return labels_1, labels_2
```

In [266]:
```
def Gmm_func(feature_1, feature_2, K):
    gmm = GaussianMixture(n_components = K)
    label_1_gmm = gmm.fit_predict(np.array(feature_1))
    label_2_gmm = gmm.fit_predict(np.array(feature_2))
    return label_1_gmm, label_2_gmm
```

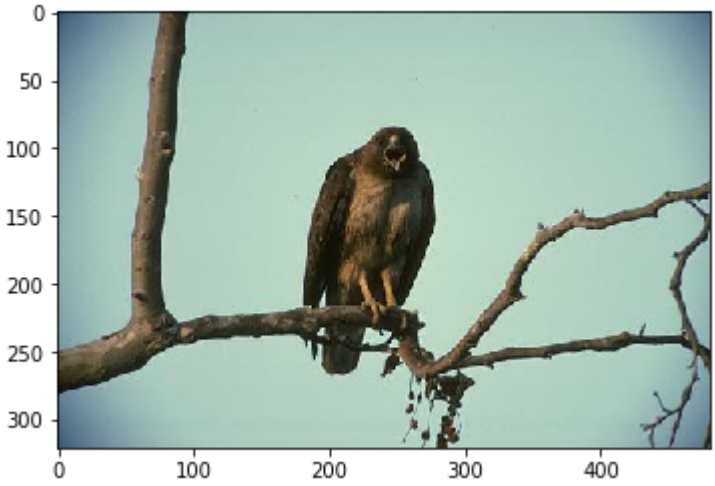
In [267]:
```
def create_pix(labels_1, labels_2, height_1, width_1, height_2, width_2, col):
    pixel_edit_1 = []
    pixel_edit_2 = []

    for i in range(height_1):
        pixel_edit_1.append([])
        for j in range(width_1):
            pixel_edit_1[i].append(col[labels_1[width_1*i + j]])

    #         if labels_1[width_1*i + j] == 1:
    #             pixel_edit_1[i].append(col[labels_1[width_1*i + j]])
    #         else:
    #             pixel_edit_1[i].append([0, 0, 0])

    for i in range(height_2):
        pixel_edit_2.append([])
        for j in range(width_2):
            pixel_edit_2[i].append(col[labels_2[width_2*i + j]])

    #         if labels_2[width_2*i + j] == 1:
    #             pixel_edit_2[i].append([250, 250, 250])
    #         else:
    #             pixel_edit_2[i].append([0, 0, 0])

    return pixel_edit_1, pixel_edit_2
```

In [268]:
```
plt.imshow(np.array(pixel_1))
```

Out[268]:
```
<matplotlib.image.AxesImage at 0x24352cd3b00>
```

In [269]:
```
plt.imshow(np.array(pixel_2))
```

Out[269]: <matplotlib.image.AxesImage at 0x24352d2ec50>



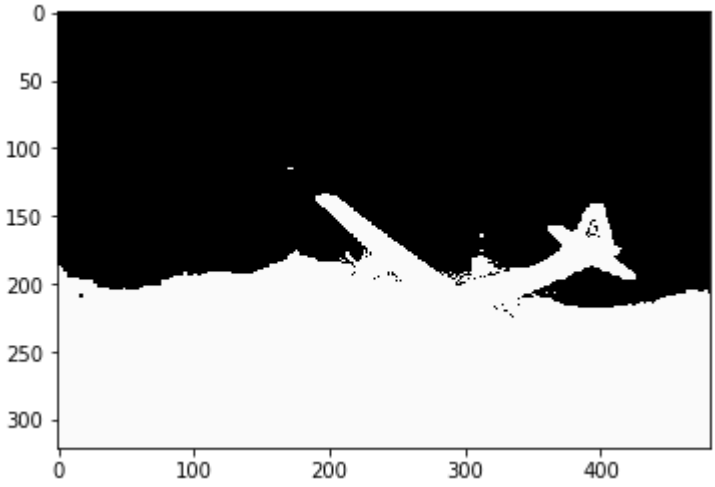
K = 2

K-means

In [270]:
```
K=2
labels_1, labels_2 = k_mean_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

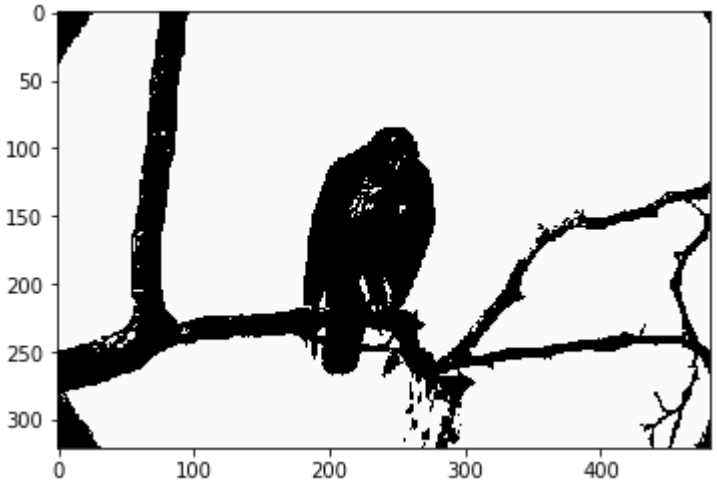
In [271]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[271]: <matplotlib.image.AxesImage at 0x243533150b8>



In [272]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[272]:
```
<matplotlib.image.AxesImage at 0x24353366ef0>
```

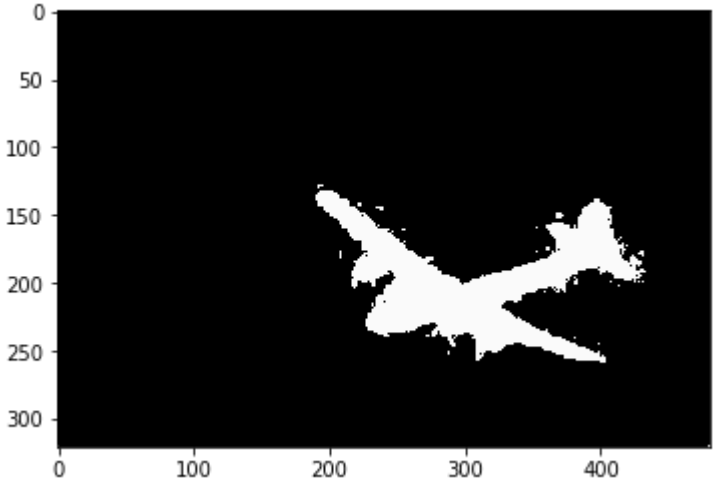


GMM

In [273]:
```
labels_1, labels_2 = Gmm_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

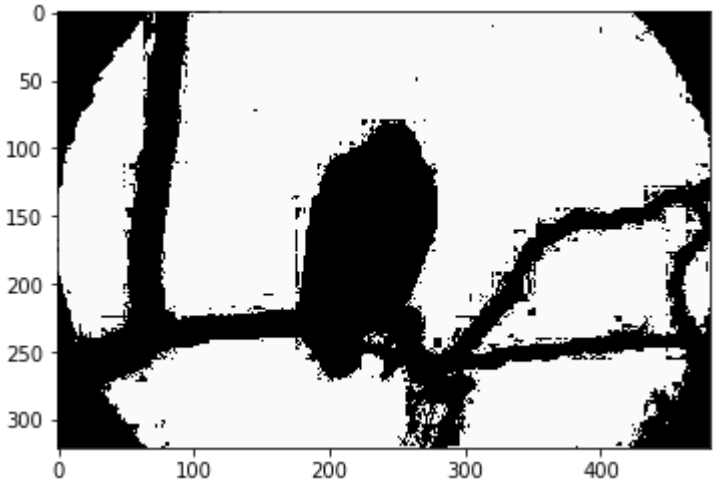
In [274]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[274]:
```
<matplotlib.image.AxesImage at 0x243533cf128>
```



In [275]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[275]:
```
<matplotlib.image.AxesImage at 0x2435342c278>
```



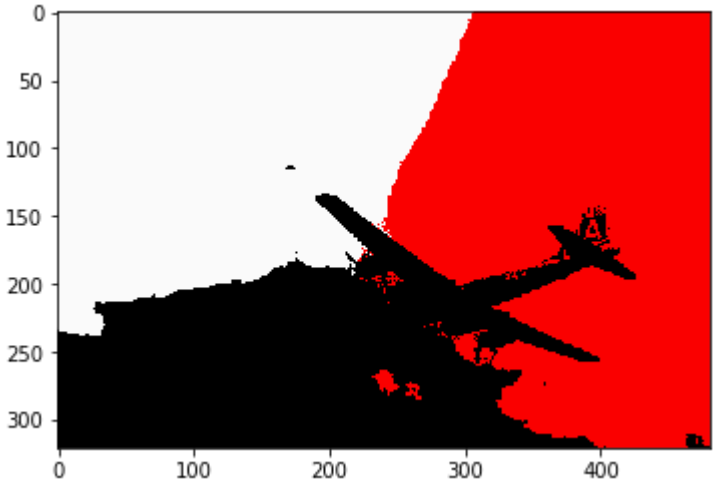
K = 3

K-means

In [276]:
```
K=3
labels_1, labels_2 = k_mean_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

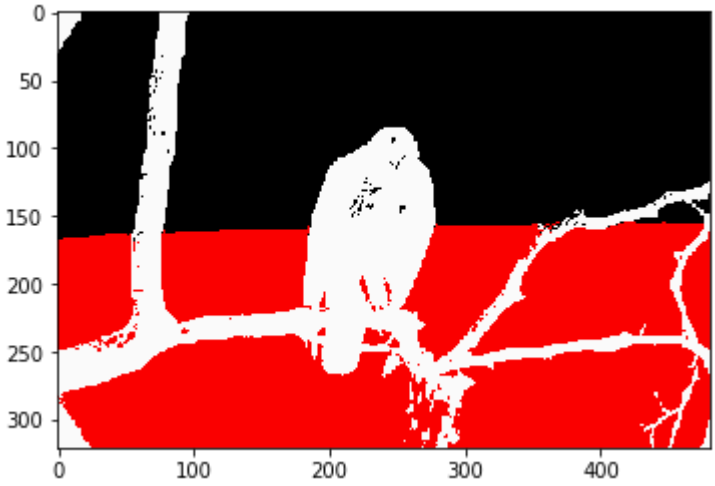
In [277]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[277]:
```
<matplotlib.image.AxesImage at 0x2435380a3c8>
```



In [278]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[278]:
```
<matplotlib.image.AxesImage at 0x243535e56a0>
```

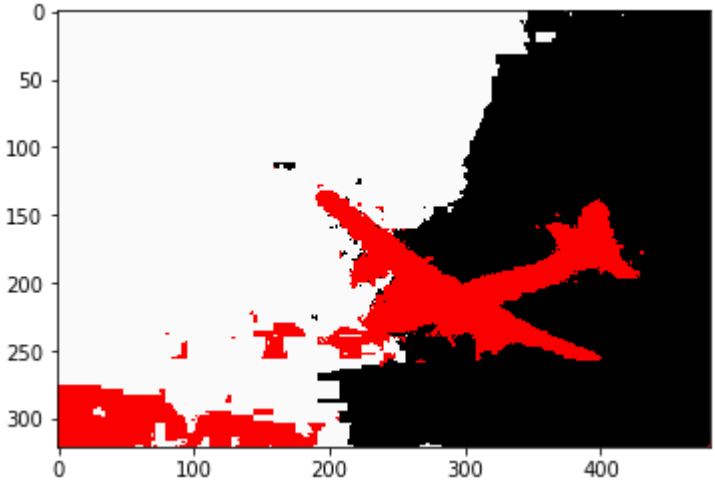


GMM

In [279]:
```
labels_1, labels_2 = Gmm_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

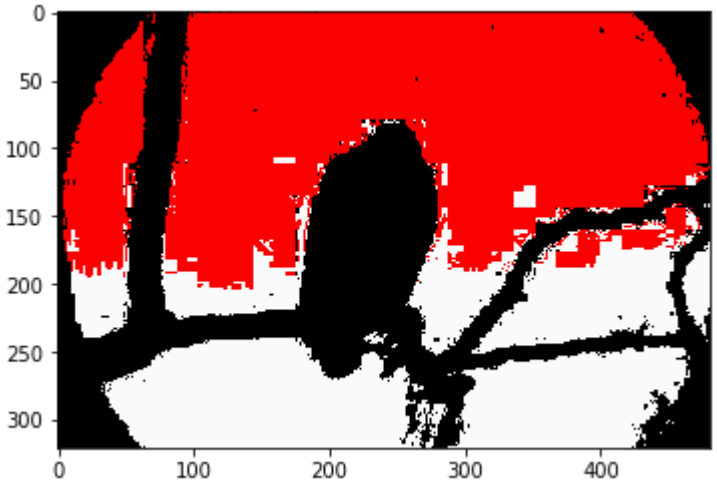
In [280]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[280]:
```
<matplotlib.image.AxesImage at 0x2435363f9e8>
```



In [281]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[281]:
```
<matplotlib.image.AxesImage at 0x2435369d908>
```



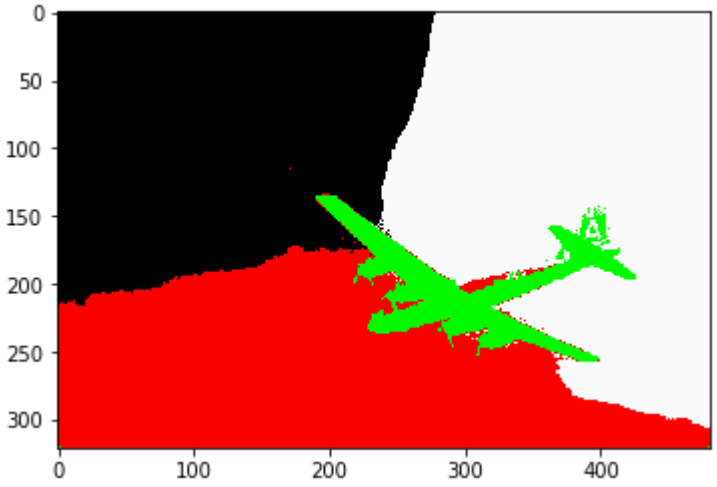
K = 4

K-means

In [282]:
```
K=4
labels_1, labels_2 = k_mean_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

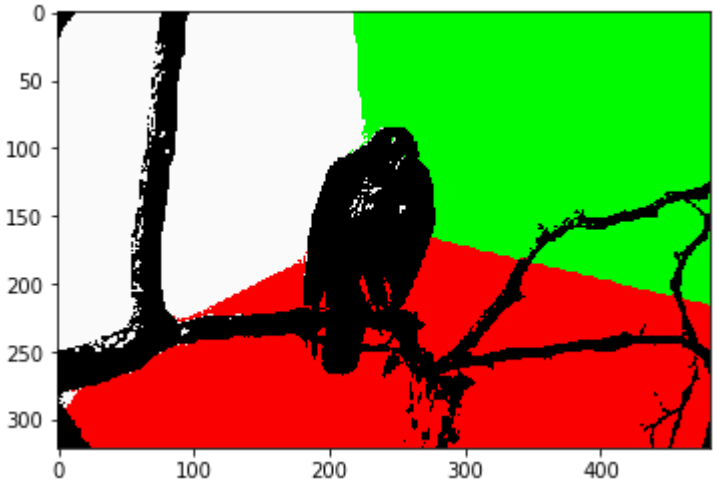
In [283]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[283]:
```
<matplotlib.image.AxesImage at 0x2435383b940>
```



In [284]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[284]:
```
<matplotlib.image.AxesImage at 0x24356149b38>
```

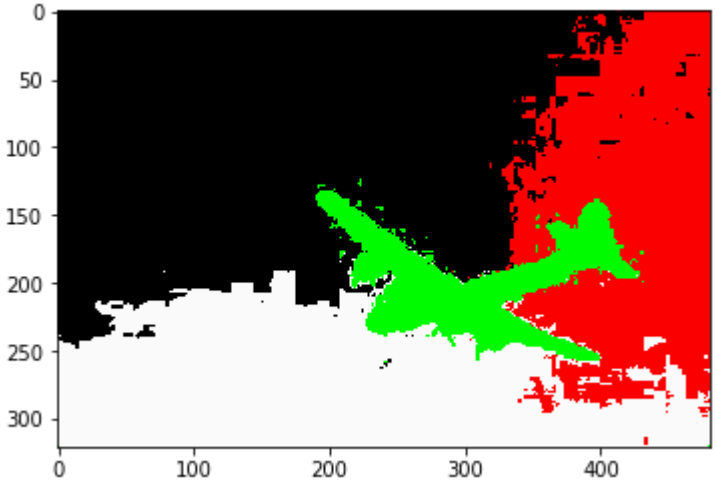


GMM

In [285]:
```
labels_1, labels_2 = Gmm_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

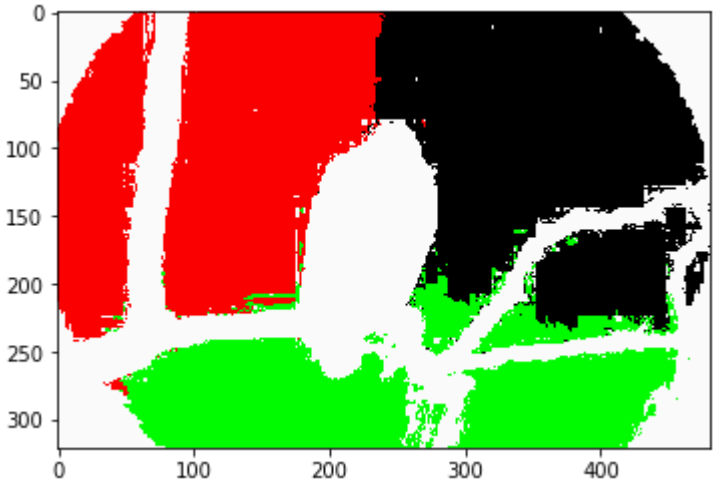
In [286]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[286]:
```
<matplotlib.image.AxesImage at 0x243561a6cf8>
```



In [287]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[287]:
```
<matplotlib.image.AxesImage at 0x243574ed080>
```



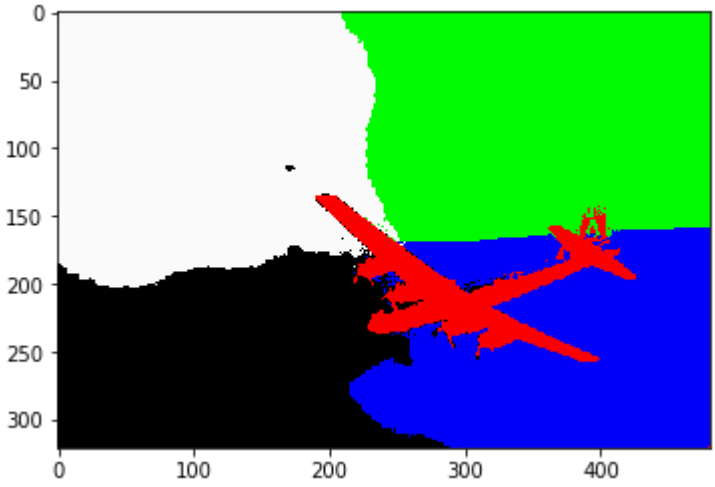
K = 5

K-means

In [288]:
```
K=5
labels_1, labels_2 = k_mean_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

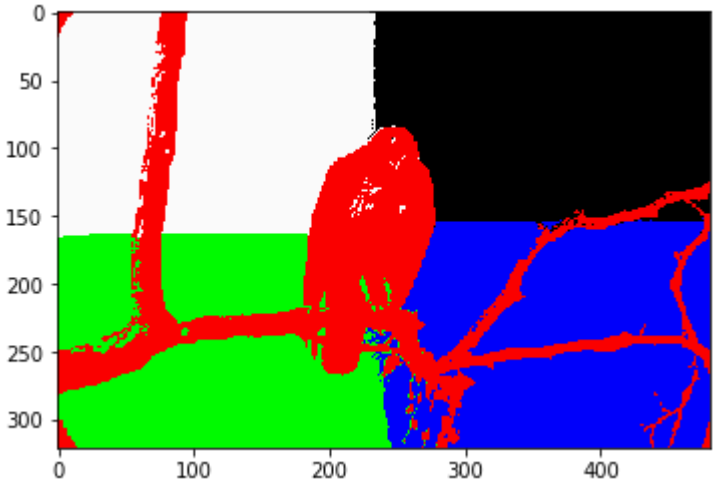
In [289]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[289]:
```
<matplotlib.image.AxesImage at 0x2435732ffd0>
```



In [290]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[290]:
```
<matplotlib.image.AxesImage at 0x24356205080>
```

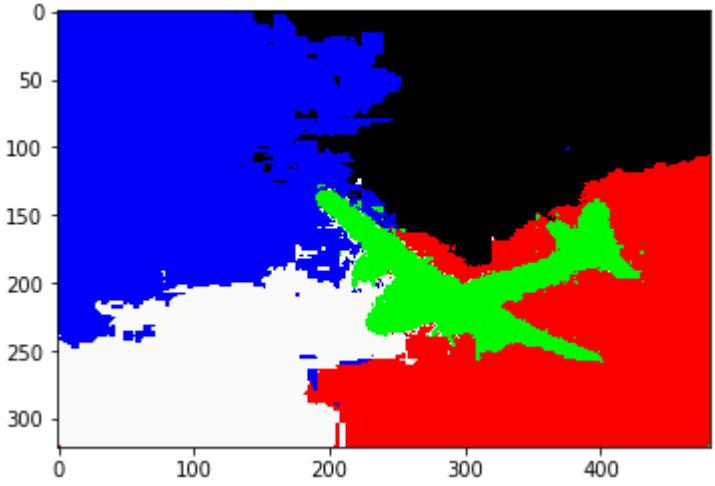


GMM

In [291]:
```
labels_1, labels_2 = Gmm_func(feature_1, feature_2, K)
pixel_edit_1, pixel_edit_2 = create_pix(labels_1, labels_2, height_1, width_1,
height_2, width_2, col)
```

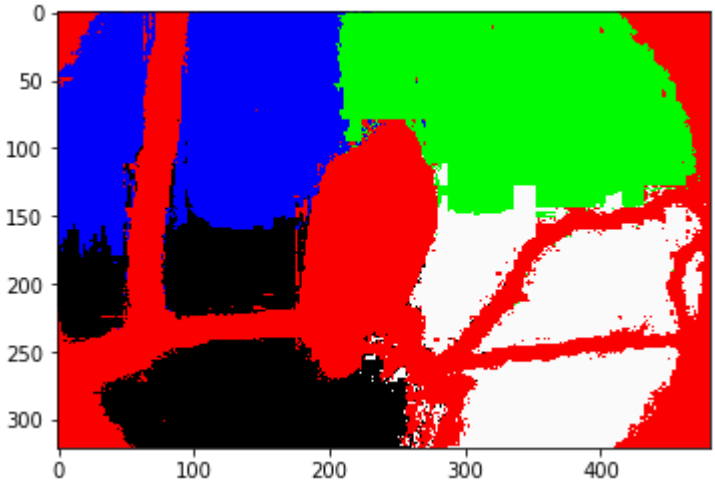
In [292]:
```
plt.imshow(np.array(pixel_edit_1))
```

Out[292]:
```
<matplotlib.image.AxesImage at 0x24356260438>
```



In [293]:
```
plt.imshow(np.array(pixel_edit_2))
```

Out[293]:
```
<matplotlib.image.AxesImage at 0x243562ba5f8>
```



Results:

1) By comparing the results in the form of the pictures, we can notice that the segmentation carried out by the K-means clustering is cleaner and the boundaries are sharper in contrast to the segmentations achieved by the GMM based clustering. This can be mainly attributed to the fact that K-means clustering works by calculating the distance of every point to the closest mean and hence points closer to a particular mean will fall into one class while the points father off will fall into some other class. On the contrary, the GMM clustering takes the likelihood of each point into consideration and not just the distance between the points. Hence, the points which necessarily might not be close to the mean of the cluster, but might be potential outliers will also be classified into appropriate classes by taking their prior probabilities and probability densities into consideration. This tends to make the segmented image seem a little less sharp.

But, as can be seen in the case of K=2, the plane is segmented quite distinctly in the GMM clustered image while it is poorly visible in the K-means segmented image. This goes to show us that even though the GMM clustering technique might not give sharp segmentation, the results are much more accurate as compared to the results of K-means, especially when the number of desired clusters does not match with the optimum working number of clusters for the particular dataset for K-means.

2) Another point to notice comes up while comparing the two segmented pictures generated by the K-means clustering method. Here, we can see that the segmentation seems quite sharp and more accurate for the picture with the bird and quite the opposite for the picture with the plane in it. This could be mainly attributed to the fact that the image with the bird has two quite contrasting colors with the sky being bright and lit while the figure of the bird is pretty dark. Thus, the K-means method can easily differentiate the two contrasting clusters which are quite distinct from each other. In the image with the plane, we notice that the colors of the plane and the sky are quite close to each other, with an added disadvantage being that the sky itself has a mix of colors in a similar range. This means that the feature points might end up being quite close to each other and hence not clearly differentiable if the criteria for clustering is the distances from the mean points of the clusters. This results in the stark difference in the accuracies of the results of the K-means clustering when compared between the two images.

This property is absent in the case of the GMM clustering method wherein even though the image might not be sharp, it is consistent for all types of images.

K-means clustering (working):

The K-means clustering works by the following method: 1) The mean centers are defined based on the number of clusters that the data is required to be divided into. The number of mean centers is equal to the number of clusters. These mean centers can be defined arbitrarily. 2) The data points are classified into clusters based on their distance to each of the mean centers. Each point is classified into the cluster whose mean is closest to it. 3) Based on the data classified into each cluster, a new mean center is calculated for each cluster. And the cycle is repeated from step 1, thus classifying data and calculating new mean centers until there is no change in classification of any of the data points from their class in the preceding iteration.

In the above code, the specific guidelines set in the question 1 of the homework have been followed. Some of the points to take note of are mentioned below: 1) The normalised 5-dimensional feature vector is given as:

[pixel_colum/total_columns, pixel_row/total_rows, R-value of pixel/max_R-value_in_image, G-value of pixel/max_G-value_in_image, B-value of pixel/max_B-value_in_image].

2) The segmentation has been visualized using different distinct colors since segmentation using gray-scales was at times not easily differentiable, especially for larger K-values (number of clusters).

GMM Clustering (working):

The GMM clustering assumes the data to be of a Gaussian Mixture Model type and performs the Estimation Maximization (EM) technique to find the optimal parameters (prior probability, mean, and covariance) for every cluster of the model. The optimal parameters are calculated using a recursive system of formulae that converge at the maximum estimation of the log-likelihood of the PDF. Here, the aim of this technique is t find clusters such that the log-likelihood of the function is maximized. This, in turn, takes the priors into consideration, thus indirectly implementing the MAP classification while classifying the data into these clusters.

In [ ]:

Solution 2:

1) Generating training dataset:

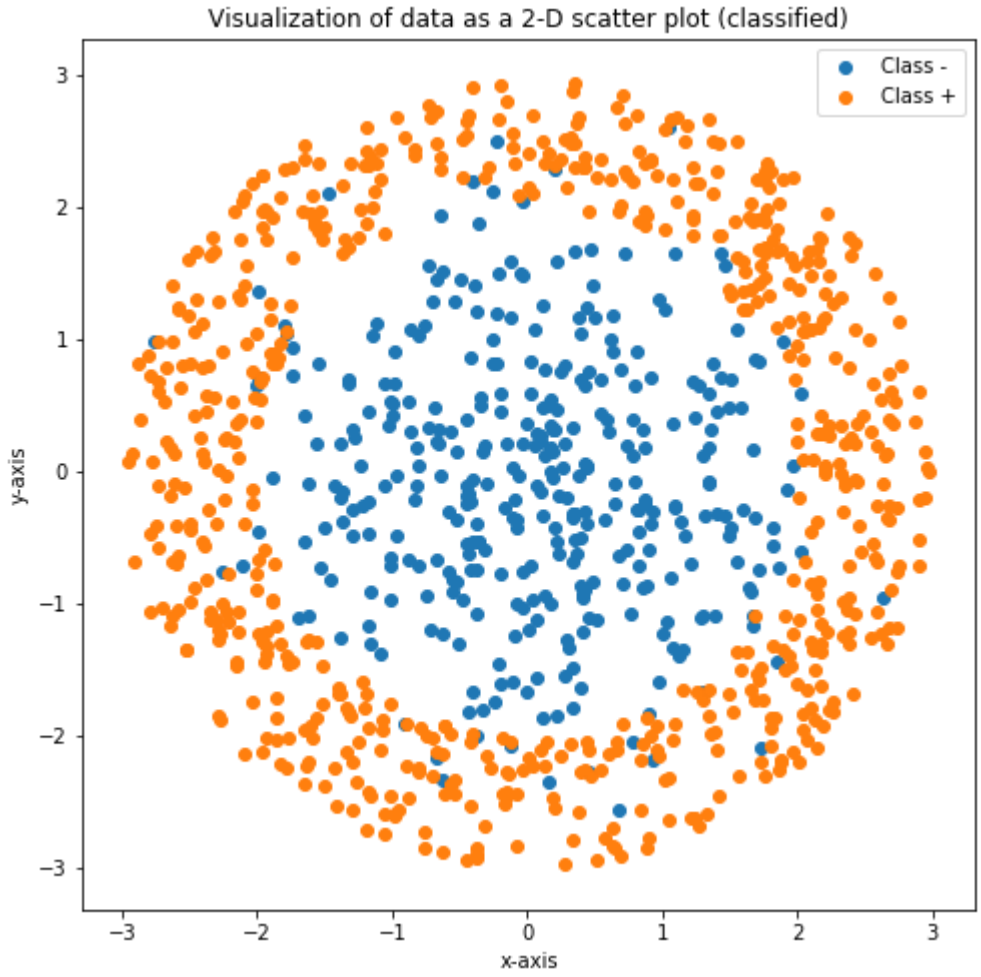
In [365]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from scipy.linalg import sqrtm
import math
from sklearn.svm import SVC
from sklearn import datasets
from sklearn.model_selection import GridSearchCV
import plotly.graph_objects as go
```

In [366]:
```
N = 1000
prior = [0.35, 0.65]
l_1 = 0
l_2 = 0

mu_x = [0, 0]
variance_x = [[1, 0], [0, 1]]

C = np.linspace(0.5, 19.5, 20)
G = np.linspace(0.25, 5, 20)
```

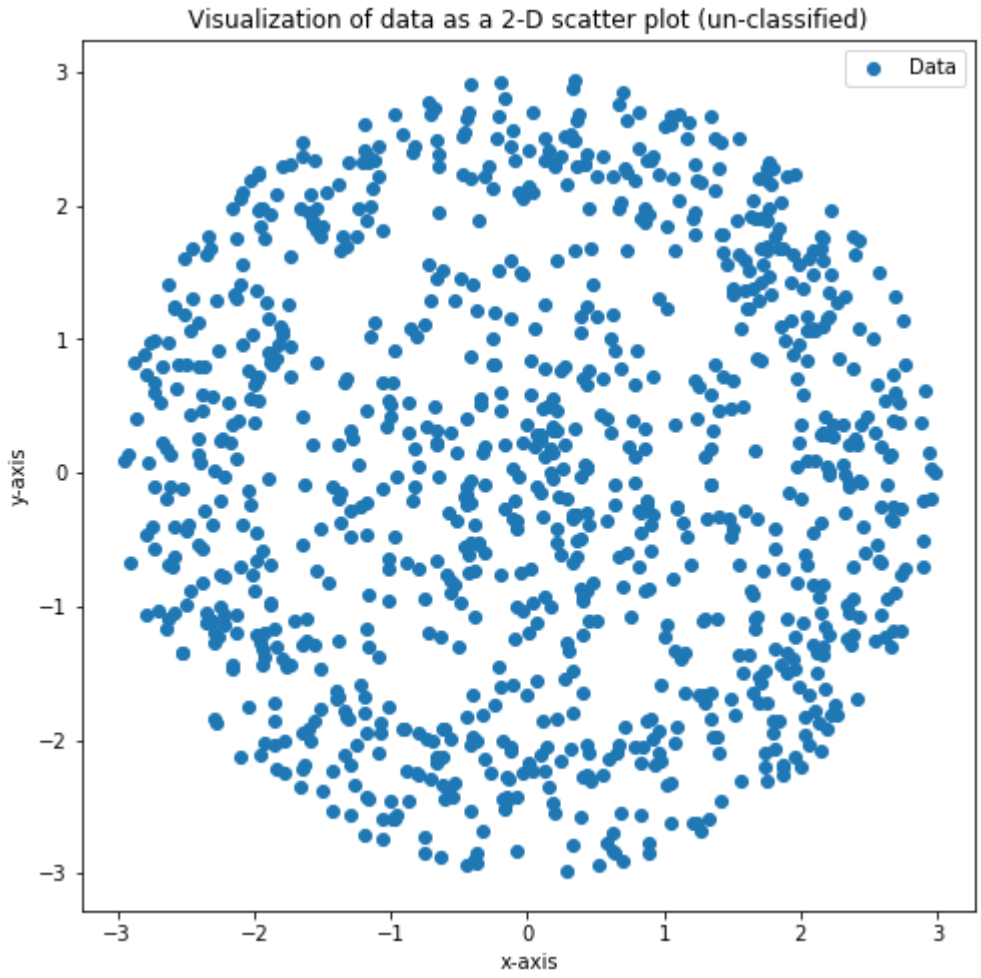
In [367]:
```
for i in range(N):
    if np.random.uniform(0, 1, 1) <= prior[0]:
        l_1 = l_1 + 1

l_2 = N - l_1
```

In [368]:
```
#Generating training dataset
data = []
data_1 = []
true_label = []
for i in range(l_1):
    temp = np.random.multivariate_normal(mu_x, variance_x, 1)
    data_1.append(temp)
    data.append(temp)
    true_label.append(0)
data_1 = np.array(data_1).reshape((l_1, 2))


data_2 = []
for i in range(l_2):
    radius = float(np.random.uniform(2, 3, 1))
    angle_rad = float(np.random.uniform(-math.pi, math.pi, 1))
    cartesian = [[radius*math.cos(angle_rad), radius*math.sin(angle_rad)]]
    data_2.append(np.array(cartesian))
    data.append(np.array(cartesian))
    true_label.append(1)
data_2 = np.array(data_2).reshape((l_2, 2))
data = np.array(data).reshape((N, 2))
```

In [369]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data_1)[:, 0], np.array(data_1)[:, 1], alpha=1, label='Cla
ss -')
ax.scatter(np.array(data_2)[:, 0], np.array(data_2)[:, 1], alpha=1, label='Cla
ss +')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Visualization of data as a 2-D scatter plot (classified)')
ax.legend()
plt.show()
```



In [370]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(data)[:, 0], np.array(data)[:, 1], alpha=1, label='Data')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Visualization of data as a 2-D scatter plot (un-classified)')
ax.legend()
plt.show()
```



2) Finding optimal hyperparameters for Linear-SVM and Gaussian-SVM

Linear-SVM

In [371]:
```
lin_accuracy = []
for c in range(len(C)):
    lin_score = []
    for i in range(10):
        new_data = []
        new_true_label = []
        test_data = []
        test_data_label = []
        for j in range(0, int(0.1*N*i), 1):
            new_data.append(data[int(j), :])
            new_true_label.append(true_label[int(j)])
        for j in range(int(0.1*N*(i+1)), N, 1):
            new_data.append(data[int(j), :])
            new_true_label.append(true_label[int(j)])
        for j in range(int(0.1*N*i), int(0.1*N*(i+1)), 1):
            test_data.append(data[int(j), :])
            test_data_label.append(true_label[int(j)])
        lin_svm = SVC(C = C[c], kernel = 'linear')
        lin_svm.fit(new_data, new_true_label)
        lin_result = lin_svm.score(test_data, test_data_label)
        lin_score.append(lin_result)
#     print(lin_score)
    lin_accuracy.append([C[c], sum(lin_score)/len(lin_score)])

lin_best = np.argmax(np.array(lin_accuracy)[:, 1])
```

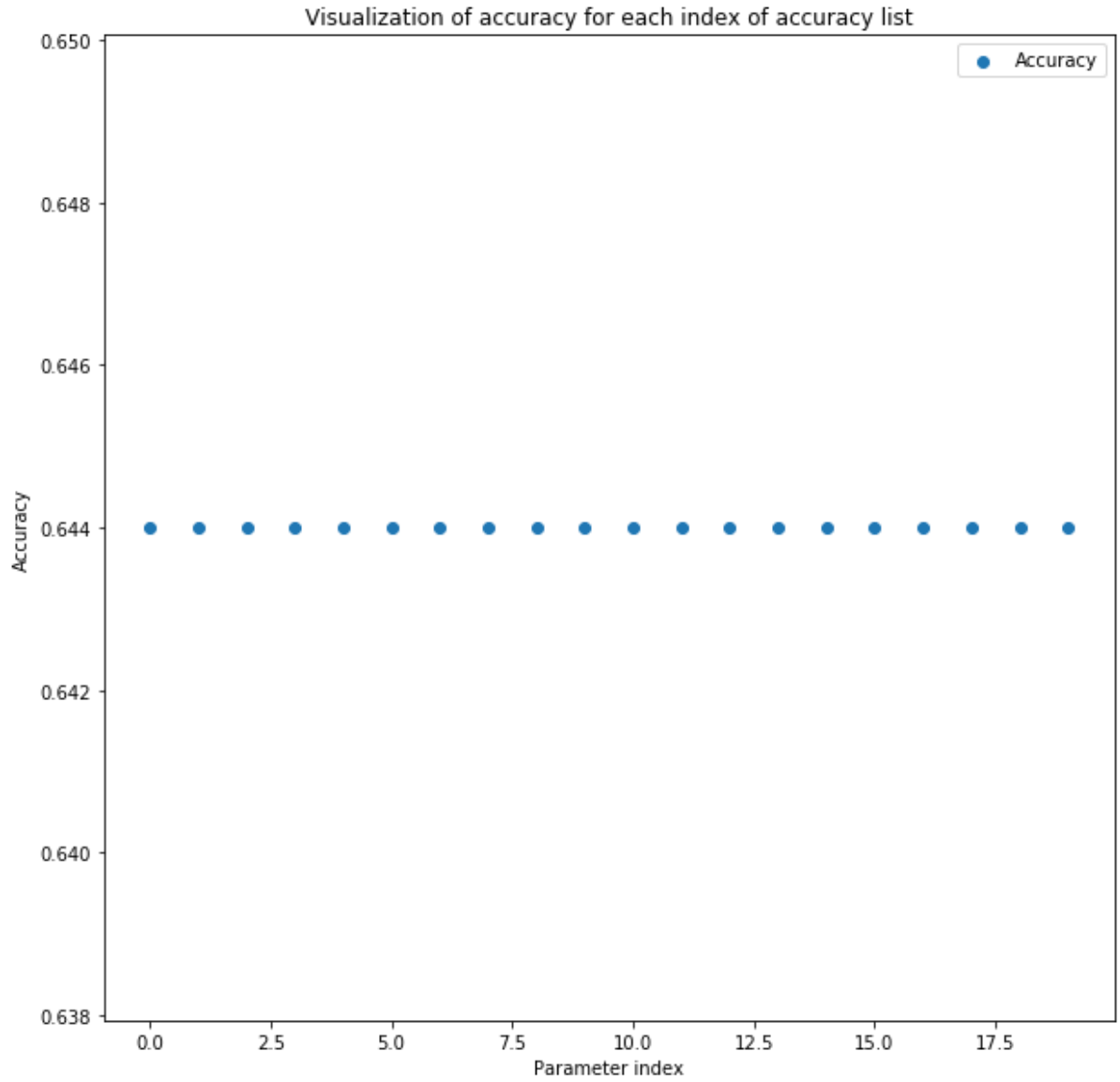
In [427]:
```
print("Index no.  \t\t C \t\t  Accuracy")
for i in range(len(lin_accuracy)):
    print("{} \t\t\t {} \t\t {}".format(i, np.array(lin_accuracy)[i, 0], np.ar
ray(lin_accuracy)[i, 1]))
```

```
Index no.                C                Accuracy
0                        0.5              0.6439999999999999
1                        1.5              0.6439999999999999
2                        2.5              0.6439999999999999
3                        3.5              0.6439999999999999
4                        4.5              0.6439999999999999
5                        5.5              0.6439999999999999
6                        6.5              0.6439999999999999
7                        7.5              0.6439999999999999
8                        8.5              0.6439999999999999
9                        9.5              0.6439999999999999
10                       10.5             0.6439999999999999
11                       11.5             0.6439999999999999
12                       12.5             0.6439999999999999
13                       13.5             0.6439999999999999
14                       14.5             0.6439999999999999
15                       15.5             0.6439999999999999
16                       16.5             0.6439999999999999
17                       17.5             0.6439999999999999
18                       18.5             0.6439999999999999
19                       19.5             0.6439999999999999
```

In [372]:
```
fig = go.Figure(data = [go.Table(header = dict(values = ['Index no.', 'C', 'Ac
curacy']), cells = dict(values = [np.arange(len(lin_accuracy)), np.array(lin_a
ccuracy)[:, 0], np.array(lin_accuracy)[:, 1]]))])
fig.show()

print("The optimal hyperparameter pair is C = {} found at index number {}".for
mat(np.array(lin_accuracy)[lin_best, 0], lin_best))
```

```
The optimal hyperparameter pair is C = 0.5 found at index number 0
```

In [373]:
```
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.arange(len(lin_accuracy)), np.array(lin_accuracy)[:, 1], label=
'Accuracy')
plt.xlabel('Parameter index')
plt.ylabel('Accuracy')
plt.title('Visualization of accuracy for each index of accuracy list')
ax.legend()
plt.show()
```



Gaussian-SVM

In [374]:
```
gaus_accuracy = []
for c in range(len(C)):
    for g in range(len(G)):
        gaus_score = []
        for i in range(10):
            new_data = []
            new_true_label = []
            test_data = []
            test_data_label = []
            for j in range(0, int(0.1*N*i), 1):
                new_data.append(data[int(j), :])
                new_true_label.append(true_label[int(j)])
            for j in range(int(0.1*N*(i+1)), N, 1):
                new_data.append(data[int(j), :])
                new_true_label.append(true_label[int(j)])
            for j in range(int(0.1*N*i), int(0.1*N*(i+1)), 1):
                test_data.append(data[int(j), :])
                test_data_label.append(true_label[int(j)])
            gaus_svm = SVC(C = C[c], kernel = 'rbf', gamma = G[g])
            gaus_svm.fit(new_data, new_true_label)
            gaus_result = gaus_svm.score(test_data, test_data_label)
            gaus_score.append(gaus_result)
#         print(gaus_score)
        gaus_accuracy.append([C[c], G[g], sum(gaus_score)/len(gaus_score)])

gaus_best = np.argmax(np.array(gaus_accuracy)[:, 2])
```

In [429]:
```
print("Index no.  \t\t C \t\t Gamma \t\t Accuracy")
for i in range(len(gaus_accuracy)):
    print("{} \t\t\t {} \t\t {} \t\t {}".format(i, np.array(gaus_accuracy)[i,
0], np.array(gaus_accuracy)[i, 1], np.array(gaus_accuracy)[i, 2]))
```

Index no.	C	Gamma	Accuracy
0	0.5	0.25	0.933
1	0.5	0.5	0.9359999999999999
2	0.5	0.75	0.9350000000000002
3	0.5	1.0	0.9359999999999999
4	0.5	1.25	0.937
5	0.5	1.5	0.9390000000000001
6	0.5	1.75	0.9390000000000001
7	0.5	2.0	0.9390000000000001
8	0.5	2.25	0.9400000000000001
9	0.5	2.5	0.9400000000000001
10	0.5	2.75	0.9410000000000001
11	0.5	3.0	0.9410000000000001
12	0.5	3.25	0.9410000000000001
13	0.5	3.5	0.942
14	0.5	3.75	0.942
15	0.5	4.0	0.942
16	0.5	4.25	0.942
17	0.5	4.5	0.942
18	0.5	4.75	0.942
19	0.5	5.0	0.942
20	1.5	0.25	0.9400000000000001
21	1.5	0.5	0.942
22	1.5	0.75	0.942
23	1.5	1.0	0.9410000000000001
24	1.5	1.25	0.942
25	1.5	1.5	0.9410000000000001
26	1.5	1.75	0.943
27	1.5	2.0	0.944
28	1.5	2.25	0.945
29	1.5	2.5	0.944
30	1.5	2.75	0.945
31	1.5	3.0	0.9480000000000001
32	1.5	3.25	0.9480000000000001
33	1.5	3.5	0.9490000000000001
34	1.5	3.75	0.9490000000000001
35	1.5	4.0	0.9490000000000001
36	1.5	4.25	0.9490000000000001
37	1.5	4.5	0.9490000000000001
38	1.5	4.75	0.95
39	1.5	5.0	0.95
40	2.5	0.25	0.944
41	2.5	0.5	0.942
42	2.5	0.75	0.942
43	2.5	1.0	0.943
44	2.5	1.25	0.945
45	2.5	1.5	0.944
46	2.5	1.75	0.944
47	2.5	2.0	0.944
48	2.5	2.25	0.9470000000000001
49	2.5	2.5	0.9480000000000001
50	2.5	2.75	0.9480000000000001
51	2.5	3.0	0.95
52	2.5	3.25	0.95
53	2.5	3.5	0.95
54	2.5	3.75	0.95
55	2.5	4.0	0.9490000000000001

56	2.5	4.25	0.9490000000000001
57	2.5	4.5	0.9490000000000001
58	2.5	4.75	0.9460000000000001
59	2.5	5.0	0.9460000000000001
60	3.5	0.25	0.944
61	3.5	0.5	0.942
62	3.5	0.75	0.942
63	3.5	1.0	0.945
64	3.5	1.25	0.945
65	3.5	1.5	0.945
66	3.5	1.75	0.9440000000000002
67	3.5	2.0	0.9460000000000001
68	3.5	2.25	0.9460000000000001
69	3.5	2.5	0.9470000000000001
70	3.5	2.75	0.95
71	3.5	3.0	0.9490000000000001
72	3.5	3.25	0.9490000000000001
73	3.5	3.5	0.9490000000000001
74	3.5	3.75	0.9470000000000001
75	3.5	4.0	0.946
76	3.5	4.25	0.944
77	3.5	4.5	0.943
78	3.5	4.75	0.944
79	3.5	5.0	0.944
80	4.5	0.25	0.944
81	4.5	0.5	0.9440000000000002
82	4.5	0.75	0.945
83	4.5	1.0	0.945
84	4.5	1.25	0.9470000000000001
85	4.5	1.5	0.9450000000000001
86	4.5	1.75	0.9450000000000001
87	4.5	2.0	0.9460000000000001
88	4.5	2.25	0.9470000000000001
89	4.5	2.5	0.9490000000000001
90	4.5	2.75	0.9490000000000001
91	4.5	3.0	0.9480000000000001
92	4.5	3.25	0.9480000000000001
93	4.5	3.5	0.9469999999999998
94	4.5	3.75	0.945
95	4.5	4.0	0.945
96	4.5	4.25	0.946
97	4.5	4.5	0.9469999999999998
98	4.5	4.75	0.9469999999999998
99	4.5	5.0	0.9469999999999998
100	5.5	0.25	0.943
101	5.5	0.5	0.9440000000000002
102	5.5	0.75	0.943
103	5.5	1.0	0.9470000000000001
104	5.5	1.25	0.9460000000000001
105	5.5	1.5	0.9450000000000001
106	5.5	1.75	0.9450000000000001
107	5.5	2.0	0.9470000000000001
108	5.5	2.25	0.9480000000000001
109	5.5	2.5	0.9480000000000001
110	5.5	2.75	0.9480000000000001
111	5.5	3.0	0.9480000000000001
112	5.5	3.25	0.9469999999999998

113	5.5	3.5	0.946
114	5.5	3.75	0.946
115	5.5	4.0	0.945
116	5.5	4.25	0.9469999999999998
117	5.5	4.5	0.9469999999999998
118	5.5	4.75	0.9480000000000001
119	5.5	5.0	0.9490000000000001
120	6.5	0.25	0.943
121	6.5	0.5	0.9470000000000001
122	6.5	0.75	0.945
123	6.5	1.0	0.9460000000000001
124	6.5	1.25	0.9460000000000001
125	6.5	1.5	0.9450000000000001
126	6.5	1.75	0.9470000000000001
127	6.5	2.0	0.9460000000000001
128	6.5	2.25	0.9480000000000001
129	6.5	2.5	0.9480000000000001
130	6.5	2.75	0.9480000000000001
131	6.5	3.0	0.9480000000000001
132	6.5	3.25	0.946
133	6.5	3.5	0.945
134	6.5	3.75	0.946
135	6.5	4.0	0.9480000000000001
136	6.5	4.25	0.9480000000000001
137	6.5	4.5	0.9490000000000001
138	6.5	4.75	0.9490000000000001
139	6.5	5.0	0.9490000000000001
140	7.5	0.25	0.945
141	7.5	0.5	0.9460000000000001
142	7.5	0.75	0.9460000000000001
143	7.5	1.0	0.9460000000000001
144	7.5	1.25	0.9450000000000001
145	7.5	1.5	0.9450000000000001
146	7.5	1.75	0.9450000000000001
147	7.5	2.0	0.9470000000000001
148	7.5	2.25	0.9490000000000001
149	7.5	2.5	0.9480000000000001
150	7.5	2.75	0.9480000000000001
151	7.5	3.0	0.9470000000000001
152	7.5	3.25	0.946
153	7.5	3.5	0.9469999999999998
154	7.5	3.75	0.9469999999999998
155	7.5	4.0	0.9490000000000001
156	7.5	4.25	0.9490000000000001
157	7.5	4.5	0.9490000000000001
158	7.5	4.75	0.9490000000000001
159	7.5	5.0	0.9490000000000001
160	8.5	0.25	0.945
161	8.5	0.5	0.9460000000000001
162	8.5	0.75	0.9460000000000001
163	8.5	1.0	0.9460000000000001
164	8.5	1.25	0.9450000000000001
165	8.5	1.5	0.9460000000000001
166	8.5	1.75	0.9450000000000001
167	8.5	2.0	0.9470000000000001
168	8.5	2.25	0.9490000000000001
169	8.5	2.5	0.9480000000000001

170	8.5	2.75	0.9469999999999998
171	8.5	3.0	0.9470000000000001
172	8.5	3.25	0.9470000000000001
173	8.5	3.5	0.9480000000000001
174	8.5	3.75	0.9490000000000001
175	8.5	4.0	0.9490000000000001
176	8.5	4.25	0.9490000000000001
177	8.5	4.5	0.9490000000000001
178	8.5	4.75	0.9490000000000001
179	8.5	5.0	0.9479999999999998
180	9.5	0.25	0.9460000000000001
181	9.5	0.5	0.945
182	9.5	0.75	0.9460000000000001
183	9.5	1.0	0.9460000000000001
184	9.5	1.25	0.9460000000000001
185	9.5	1.5	0.9460000000000001
186	9.5	1.75	0.945
187	9.5	2.0	0.9470000000000001
188	9.5	2.25	0.9490000000000001
189	9.5	2.5	0.9469999999999998
190	9.5	2.75	0.9460000000000001
191	9.5	3.0	0.9470000000000001
192	9.5	3.25	0.9480000000000001
193	9.5	3.5	0.9490000000000001
194	9.5	3.75	0.9490000000000001
195	9.5	4.0	0.9479999999999998
196	9.5	4.25	0.9490000000000001
197	9.5	4.5	0.9490000000000001
198	9.5	4.75	0.9479999999999998
199	9.5	5.0	0.9480000000000001
200	10.5	0.25	0.9470000000000001
201	10.5	0.5	0.945
202	10.5	0.75	0.9470000000000001
203	10.5	1.0	0.9460000000000001
204	10.5	1.25	0.9460000000000001
205	10.5	1.5	0.945
206	10.5	1.75	0.9460000000000001
207	10.5	2.0	0.9490000000000001
208	10.5	2.25	0.9480000000000001
209	10.5	2.5	0.9469999999999998
210	10.5	2.75	0.946
211	10.5	3.0	0.9480000000000001
212	10.5	3.25	0.9490000000000001
213	10.5	3.5	0.9490000000000001
214	10.5	3.75	0.9479999999999998
215	10.5	4.0	0.9479999999999998
216	10.5	4.25	0.9490000000000001
217	10.5	4.5	0.9479999999999998
218	10.5	4.75	0.9490000000000001
219	10.5	5.0	0.9480000000000001
220	11.5	0.25	0.9460000000000001
221	11.5	0.5	0.945
222	11.5	0.75	0.9460000000000001
223	11.5	1.0	0.9480000000000001
224	11.5	1.25	0.9450000000000001
225	11.5	1.5	0.945
226	11.5	1.75	0.9469999999999998

227	11.5	2.0	0.9490000000000001
228	11.5	2.25	0.9469999999999998
229	11.5	2.5	0.9460000000000001
230	11.5	2.75	0.946
231	11.5	3.0	0.9490000000000001
232	11.5	3.25	0.95
233	11.5	3.5	0.9479999999999998
234	11.5	3.75	0.9479999999999998
235	11.5	4.0	0.9479999999999998
236	11.5	4.25	0.9479999999999998
237	11.5	4.5	0.9479999999999998
238	11.5	4.75	0.9480000000000001
239	11.5	5.0	0.9470000000000001
240	12.5	0.25	0.9460000000000001
241	12.5	0.5	0.9470000000000001
242	12.5	0.75	0.9460000000000001
243	12.5	1.0	0.9470000000000001
244	12.5	1.25	0.9450000000000001
245	12.5	1.5	0.945
246	12.5	1.75	0.9469999999999998
247	12.5	2.0	0.9469999999999998
248	12.5	2.25	0.9469999999999998
249	12.5	2.5	0.9460000000000001
250	12.5	2.75	0.9469999999999998
251	12.5	3.0	0.9490000000000001
252	12.5	3.25	0.9489999999999998
253	12.5	3.5	0.9479999999999998
254	12.5	3.75	0.9479999999999998
255	12.5	4.0	0.9479999999999998
256	12.5	4.25	0.9479999999999998
257	12.5	4.5	0.9490000000000001
258	12.5	4.75	0.9470000000000001
259	12.5	5.0	0.9470000000000001
260	13.5	0.25	0.9460000000000001
261	13.5	0.5	0.9470000000000001
262	13.5	0.75	0.9470000000000001
263	13.5	1.0	0.945
264	13.5	1.25	0.9450000000000001
265	13.5	1.5	0.9469999999999998
266	13.5	1.75	0.9469999999999998
267	13.5	2.0	0.9469999999999998
268	13.5	2.25	0.9469999999999998
269	13.5	2.5	0.9469999999999998
270	13.5	2.75	0.9480000000000001
271	13.5	3.0	0.95
272	13.5	3.25	0.9489999999999998
273	13.5	3.5	0.9479999999999998
274	13.5	3.75	0.9479999999999998
275	13.5	4.0	0.9479999999999998
276	13.5	4.25	0.9480000000000001
277	13.5	4.5	0.9470000000000001
278	13.5	4.75	0.9470000000000001
279	13.5	5.0	0.9460000000000001
280	14.5	0.25	0.9460000000000001
281	14.5	0.5	0.9470000000000001
282	14.5	0.75	0.9470000000000001
283	14.5	1.0	0.9460000000000001

284	14.5	1.25	0.9450000000000001
285	14.5	1.5	0.9469999999999998
286	14.5	1.75	0.9460000000000001
287	14.5	2.0	0.9469999999999998
288	14.5	2.25	0.9469999999999998
289	14.5	2.5	0.946
290	14.5	2.75	0.9480000000000001
291	14.5	3.0	0.9489999999999998
292	14.5	3.25	0.9479999999999998
293	14.5	3.5	0.9479999999999998
294	14.5	3.75	0.9479999999999998
295	14.5	4.0	0.9469999999999998
296	14.5	4.25	0.9469999999999998
297	14.5	4.5	0.9470000000000001
298	14.5	4.75	0.9460000000000001
299	14.5	5.0	0.9470000000000001
300	15.5	0.25	0.9460000000000001
301	15.5	0.5	0.9470000000000001
302	15.5	0.75	0.9470000000000001
303	15.5	1.0	0.9460000000000001
304	15.5	1.25	0.9460000000000001
305	15.5	1.5	0.9469999999999998
306	15.5	1.75	0.9469999999999998
307	15.5	2.0	0.9469999999999998
308	15.5	2.25	0.9469999999999998
309	15.5	2.5	0.9480000000000001
310	15.5	2.75	0.9480000000000001
311	15.5	3.0	0.9489999999999998
312	15.5	3.25	0.9479999999999998
313	15.5	3.5	0.9479999999999998
314	15.5	3.75	0.9479999999999998
315	15.5	4.0	0.9469999999999998
316	15.5	4.25	0.9470000000000001
317	15.5	4.5	0.9480000000000001
318	15.5	4.75	0.9460000000000001
319	15.5	5.0	0.9470000000000001
320	16.5	0.25	0.9460000000000001
321	16.5	0.5	0.9470000000000001
322	16.5	0.75	0.9470000000000001
323	16.5	1.0	0.9460000000000001
324	16.5	1.25	0.9470000000000001
325	16.5	1.5	0.9469999999999998
326	16.5	1.75	0.9469999999999998
327	16.5	2.0	0.9469999999999998
328	16.5	2.25	0.9469999999999998
329	16.5	2.5	0.9480000000000001
330	16.5	2.75	0.9480000000000001
331	16.5	3.0	0.9489999999999998
332	16.5	3.25	0.9479999999999998
333	16.5	3.5	0.9479999999999998
334	16.5	3.75	0.9469999999999998
335	16.5	4.0	0.9480000000000001
336	16.5	4.25	0.9480000000000001
337	16.5	4.5	0.9470000000000001
338	16.5	4.75	0.9470000000000001
339	16.5	5.0	0.9460000000000001
340	17.5	0.25	0.945

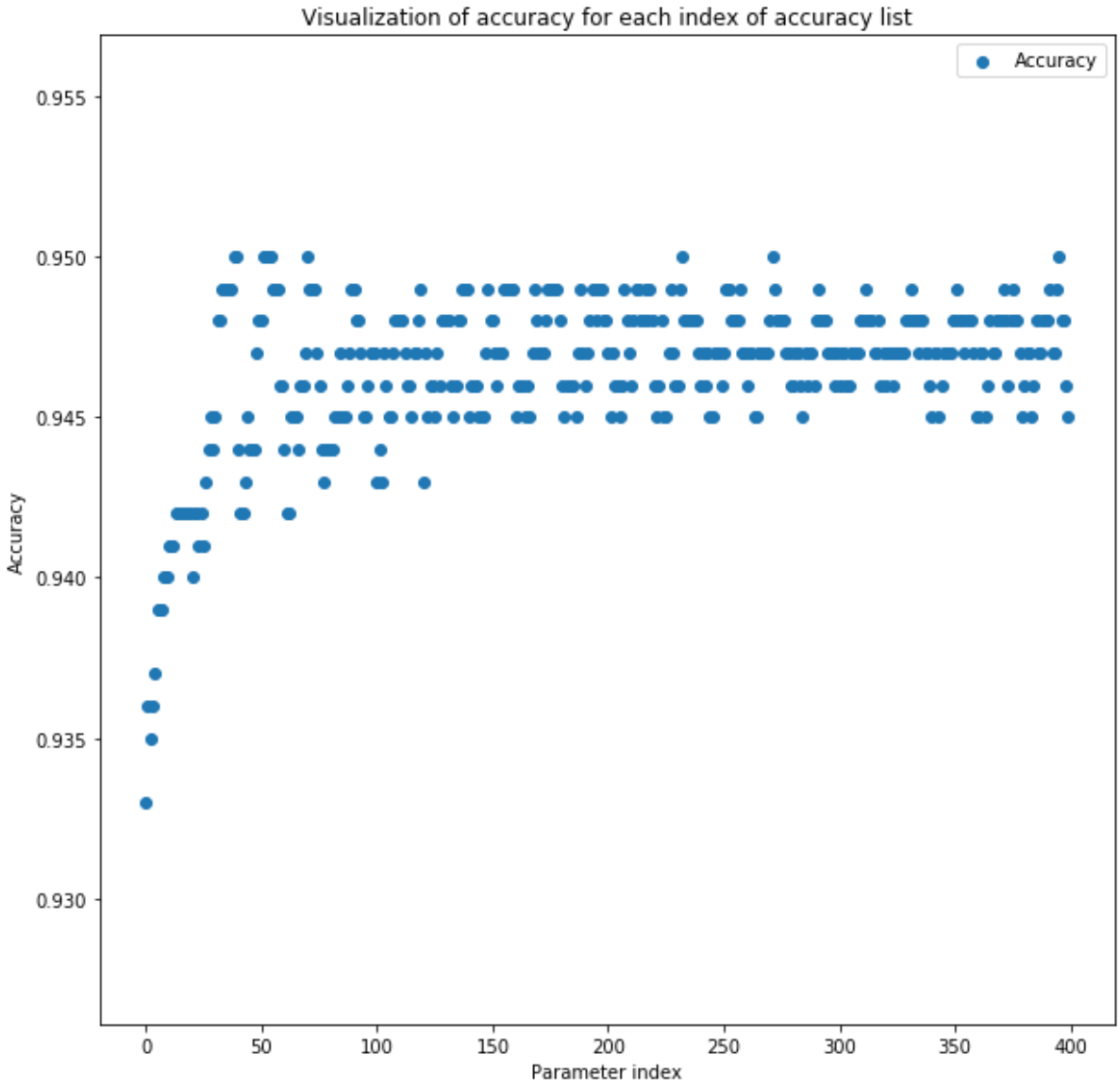
341	17.5	0.5	0.9470000000000001
342	17.5	0.75	0.9470000000000001
343	17.5	1.0	0.9450000000000001
344	17.5	1.25	0.9460000000000001
345	17.5	1.5	0.9469999999999998
346	17.5	1.75	0.9469999999999998
347	17.5	2.0	0.9469999999999998
348	17.5	2.25	0.9469999999999998
349	17.5	2.5	0.9480000000000001
350	17.5	2.75	0.9480000000000001
351	17.5	3.0	0.9489999999999998
352	17.5	3.25	0.9479999999999998
353	17.5	3.5	0.9479999999999998
354	17.5	3.75	0.9469999999999998
355	17.5	4.0	0.9480000000000001
356	17.5	4.25	0.9480000000000001
357	17.5	4.5	0.9480000000000001
358	17.5	4.75	0.9470000000000001
359	17.5	5.0	0.945
360	18.5	0.25	0.945
361	18.5	0.5	0.9470000000000001
362	18.5	0.75	0.9470000000000001
363	18.5	1.0	0.9450000000000001
364	18.5	1.25	0.9460000000000001
365	18.5	1.5	0.9480000000000001
366	18.5	1.75	0.9469999999999998
367	18.5	2.0	0.9469999999999998
368	18.5	2.25	0.9480000000000001
369	18.5	2.5	0.9480000000000001
370	18.5	2.75	0.9480000000000001
371	18.5	3.0	0.9489999999999998
372	18.5	3.25	0.9479999999999998
373	18.5	3.5	0.946
374	18.5	3.75	0.9480000000000001
375	18.5	4.0	0.9490000000000001
376	18.5	4.25	0.9480000000000001
377	18.5	4.5	0.9480000000000001
378	18.5	4.75	0.9469999999999998
379	18.5	5.0	0.945
380	19.5	0.25	0.9460000000000001
381	19.5	0.5	0.9470000000000001
382	19.5	0.75	0.9470000000000001
383	19.5	1.0	0.9450000000000001
384	19.5	1.25	0.9460000000000001
385	19.5	1.5	0.9480000000000001
386	19.5	1.75	0.9469999999999998
387	19.5	2.0	0.9469999999999998
388	19.5	2.25	0.9480000000000001
389	19.5	2.5	0.9480000000000001
390	19.5	2.75	0.9480000000000001
391	19.5	3.0	0.9489999999999998
392	19.5	3.25	0.9469999999999998
393	19.5	3.5	0.9470000000000001
394	19.5	3.75	0.9490000000000001
395	19.5	4.0	0.95
396	19.5	4.25	0.9480000000000001
397	19.5	4.5	0.9480000000000001

398	19.5	4.75	0.946
399	19.5	5.0	0.9450000000000001

In [375]:
```
fig = go.Figure(data = [go.Table(header = dict(values = ['Index no.', 'C', 'Ga
mma', 'Accuracy']), cells = dict(values = [np.arange(len(gaus_accuracy)), np.a
rray(gaus_accuracy)[:, 0], np.array(gaus_accuracy)[:, 1], np.array(gaus_accura
cy)[:, 2]]))])
fig.show()

print("The optimal hyperparameter pair is C = {} and Gamma = {} found at index
number {}".format(np.array(gaus_accuracy)[gaus_best, 0], np.array(gaus_accurac
y)[gaus_best, 1], gaus_best))
```

```
The optimal hyperparameter pair is C = 1.5 and Gamma = 4.75 found at index nu
mber 38
```

In [376]:
```
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.arange(len(gaus_accuracy)), np.array(gaus_accuracy)[:, 2], label
='Accuracy')
plt.xlabel('Parameter index')
plt.ylabel('Accuracy')
plt.title('Visualization of accuracy for each index of accuracy list')
ax.legend()
plt.show()
```



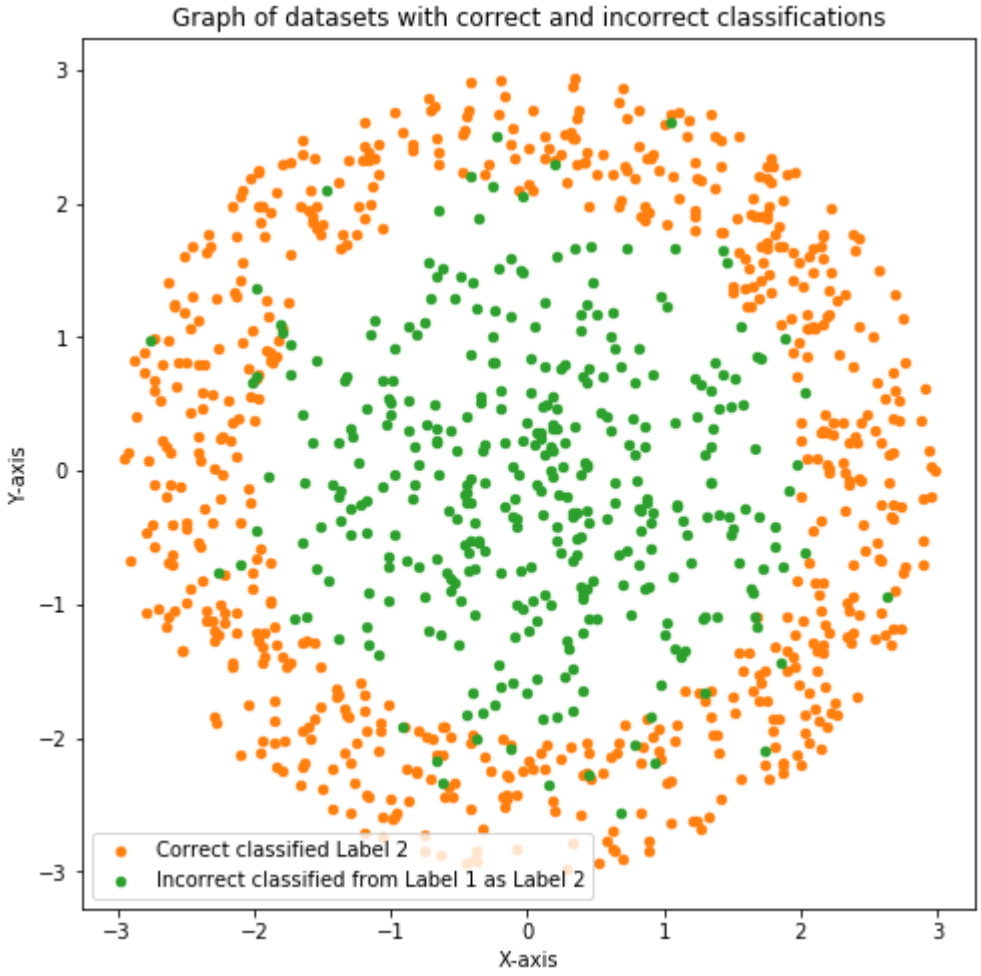
3) Classification of entire training dataset using optimal hyperparameters

Linear-SVM

In [377]:
```
lin_svm = SVC(C = np.array(lin_accuracy)[lin_best, 0], kernel = 'linear')
lin_svm.fit(data, true_label)
lin_label = lin_svm.predict(data)
lin_score = lin_svm.score(data, true_label)
```

In [378]:
```
lin_error_1 = []
lin_right_1 = []
lin_error_2 = []
lin_right_2 = []

for i in range(l_1):
    if lin_label[i] == 1:
        lin_error_1.append(data[i])
    else:
        lin_right_1.append(data[i])

for i in range(l_2):
    if lin_label[l_1+i] == 0:
        lin_error_2.append(data[l_1+i])
    else:
        lin_right_2.append(data[l_1+i])
```

In [379]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
if len(lin_right_1) > 0:
    ax.scatter(np.array(lin_right_1)[:, 0], np.array(lin_right_1)[:, 1], s=20,
color='tab:blue', alpha=1, label='Correct classified Label 1')
if len(lin_right_2) > 0:
    ax.scatter(np.array(lin_right_2)[:, 0], np.array(lin_right_2)[:, 1], s=20,
color='tab:orange', alpha=1, label='Correct classified Label 2')
if len(lin_error_1) > 0:
    ax.scatter(np.array(lin_error_1)[:, 0], np.array(lin_error_1)[:, 1], s=20,
color='tab:green', alpha=1, label='Incorrect classified from Label 1 as Label
 2')
if len(lin_error_2) > 0:
    ax.scatter(np.array(lin_error_2)[:, 0], np.array(lin_error_2)[:, 1], s=20,
color='tab:red', alpha=1, label='Incorrect classified from Label 2 as Label 1'
)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of datasets with correct and incorrect classifications')
ax.legend()
plt.show()
```



In [380]:
```
print("The accuracy is: {}".format(lin_score))
print("The total number of errors are: {}".format(len(lin_error_1) + len(lin_e
rror_2)))
print("The error probability is {}.".format((len(lin_error_1) + len(lin_error_
2))/N))
```

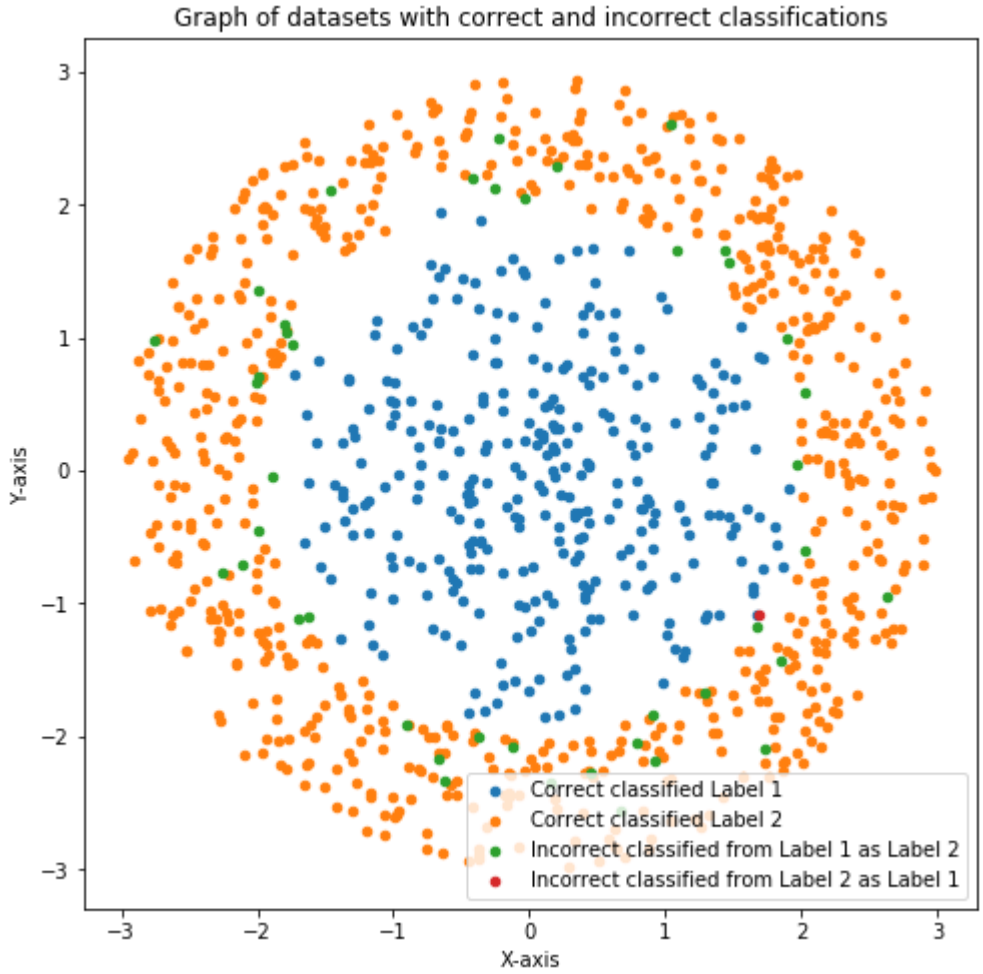
```
The accuracy is: 0.644
The total number of errors are: 356
The error probability is 0.356.
```

Gaussian-SVM

In [381]:
```
gaus_svm = SVC(C = np.array(gaus_accuracy)[gaus_best, 0], kernel = 'rbf', gamm
a = np.array(gaus_accuracy)[gaus_best, 1])
gaus_svm.fit(data, true_label)
gaus_label = gaus_svm.predict(data)
gaus_score = gaus_svm.score(data, true_label)
```

In [382]:
```
gaus_error_1 = []
gaus_right_1 = []
gaus_error_2 = []
gaus_right_2 = []

for i in range(l_1):
    if gaus_label[i] == 1:
        gaus_error_1.append(data[i])
    else:
        gaus_right_1.append(data[i])

for i in range(l_2):
    if gaus_label[l_1+i] == 0:
        gaus_error_2.append(data[l_1+i])
    else:
        gaus_right_2.append(data[l_1+i])
```

In [383]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
if len(gaus_right_1) > 0:
    ax.scatter(np.array(gaus_right_1)[:, 0], np.array(gaus_right_1)[:, 1], s=2
0, color='tab:blue', alpha=1, label='Correct classified Label 1')
if len(gaus_right_2) > 0:
    ax.scatter(np.array(gaus_right_2)[:, 0], np.array(gaus_right_2)[:, 1], s=2
0, color='tab:orange', alpha=1, label='Correct classified Label 2')
if len(gaus_error_1) > 0:
    ax.scatter(np.array(gaus_error_1)[:, 0], np.array(gaus_error_1)[:, 1], s=2
0, color='tab:green', alpha=1, label='Incorrect classified from Label 1 as Lab
el 2')
if len(gaus_error_2) > 0:
    ax.scatter(np.array(gaus_error_2)[:, 0], np.array(gaus_error_2)[:, 1], s=2
0, color='tab:red', alpha=1, label='Incorrect classified from Label 2 as Label
1')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of datasets with correct and incorrect classifications')
ax.legend()
plt.show()
```



In [384]:
```
print("The accuracy is: {}".format(gaus_score))
print("The total number of errors are: {}".format(len(gaus_error_1) + len(gaus
_error_2)))
print("The error probability is {}.".format((len(gaus_error_1) + len(gaus_erro
r_2))/N))
```

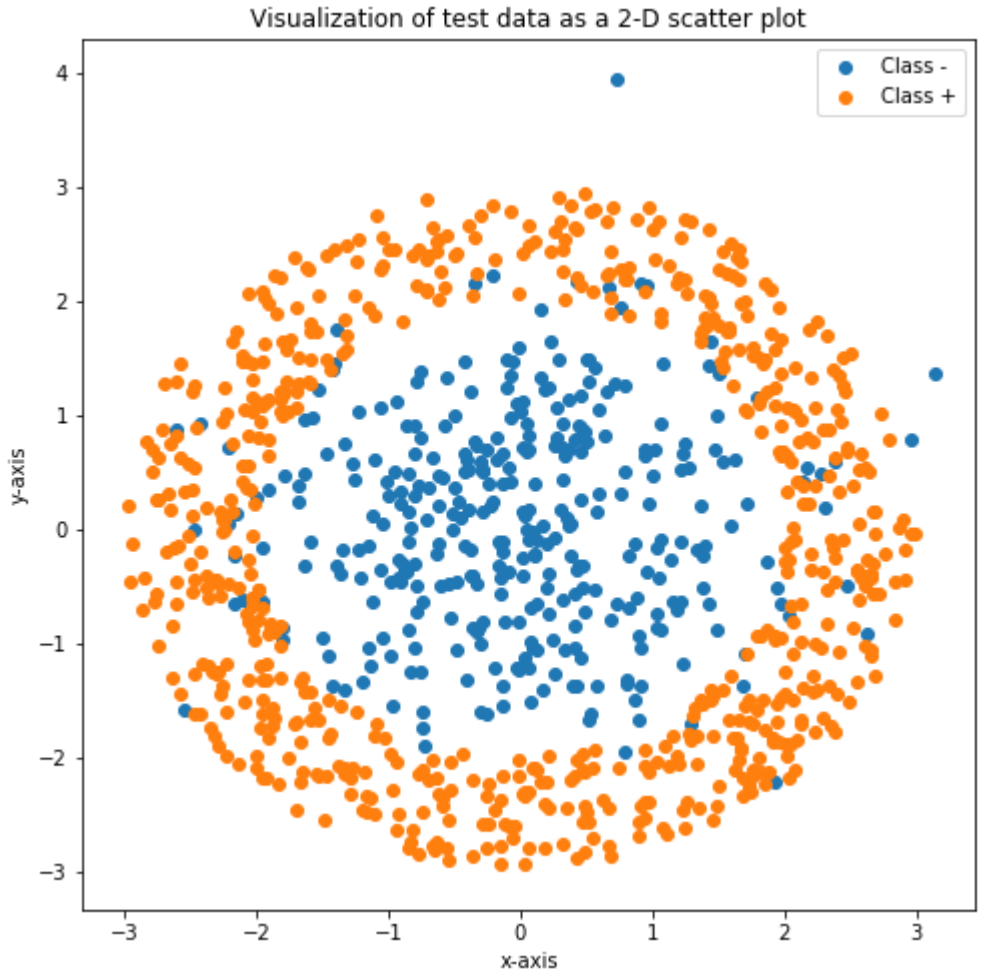
```
The accuracy is: 0.956
The total number of errors are: 44
The error probability is 0.044.
```

4) Classification of test dataset on trained classifier

Generating test dataset

In [415]:
```
#Generating test data for part 4 of the question

test_N = 1000
test_l_1 = 0
test_l_2 = 0

for i in range(test_N):
    if np.random.uniform(0, 1, 1) <= prior[0]:
        test_l_1 = test_l_1 + 1
test_l_2 = test_N - test_l_1

test_samples_1 = []
test_samples = []
test_true_label = []
for i in range(test_l_1):
    temp = np.random.multivariate_normal(mu_x, variance_x, 1)
    test_samples_1.append(temp)
    test_samples.append(temp)
    test_true_label.append(0)
test_samples_1 = np.array(test_samples_1).reshape((test_l_1, 2))

test_samples_2 = []
for i in range(test_l_2):
    radius = float(np.random.uniform(2, 3, 1))
    angle_rad = float(np.random.uniform(-math.pi, math.pi, 1))
    cartesian = [[radius*math.cos(angle_rad), radius*math.sin(angle_rad)]]
    test_samples_2.append(np.array(cartesian))
    test_samples.append(np.array(cartesian))
    test_true_label.append(1)
test_samples_2 = np.array(test_samples_2).reshape((test_l_2, 2))
test_samples = np.array(test_samples).reshape((test_N, 2))
```

In [416]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
ax.scatter(np.array(test_samples_1)[:, 0], np.array(test_samples_1)[:, 1], alp
ha=1, label='Class -')
ax.scatter(np.array(test_samples_2)[:, 0], np.array(test_samples_2)[:, 1], alp
ha=1, label='Class +')
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.title('Visualization of test data as a 2-D scatter plot')
ax.legend()
plt.show()
```

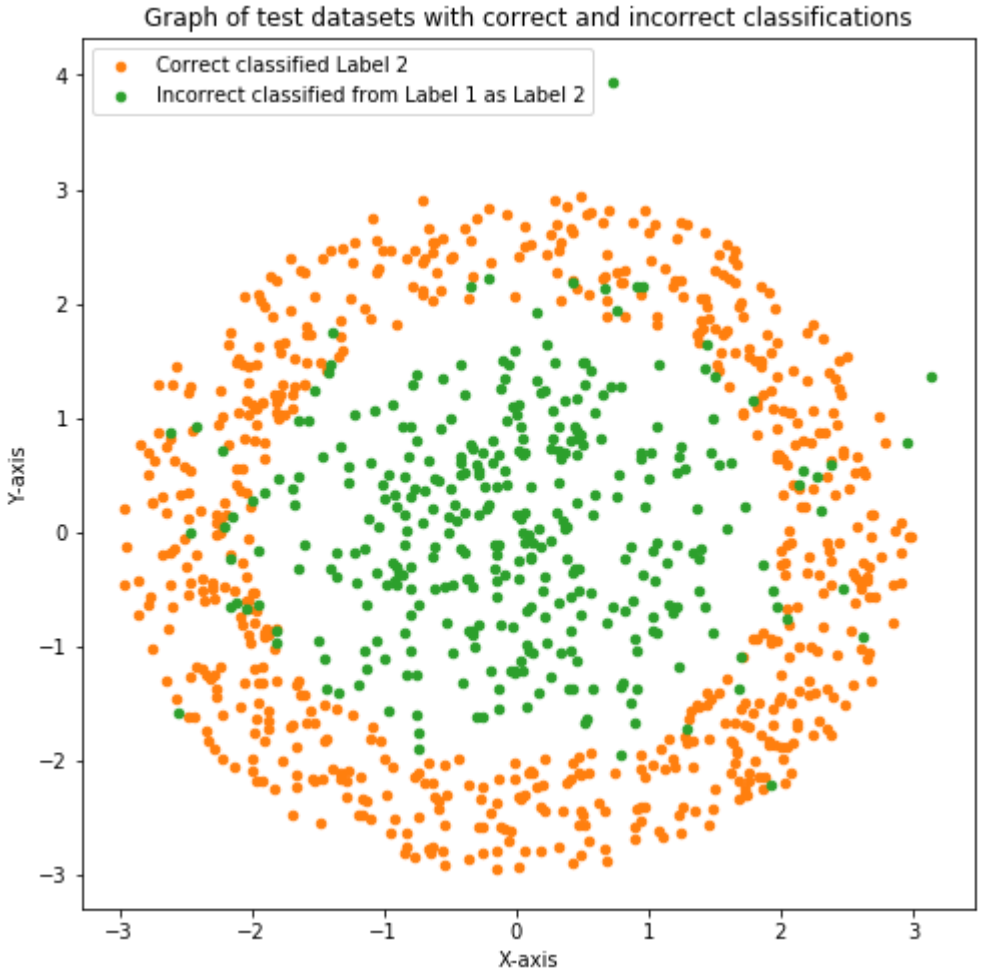


Linear-SVM

In [417]:
```
test_lin_label = lin_svm.predict(test_samples)
test_lin_score = lin_svm.score(test_samples, test_true_label)
```

In [418]:
```
test_lin_error_1 = []
test_lin_right_1 = []
test_lin_error_2 = []
test_lin_right_2 = []

for i in range(test_l_1):
    if test_lin_label[i] == 1:
        test_lin_error_1.append(test_samples[i])
    else:
        test_lin_right_1.append(test_samples[i])

for i in range(test_l_2):
    if test_lin_label[test_l_1+i] == 0:
        test_lin_error_2.append(test_samples[test_l_1+i])
    else:
        test_lin_right_2.append(test_samples[test_l_1+i])
```

In [419]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
if len(test_lin_right_1) > 0:
    ax.scatter(np.array(test_lin_right_1)[:, 0], np.array(test_lin_right_1)[:,
1], s=20, color='tab:blue', alpha=1, label='Correct classified Label 1')
if len(test_lin_right_2) > 0:
    ax.scatter(np.array(test_lin_right_2)[:, 0], np.array(test_lin_right_2)[:,
1], s=20, color='tab:orange', alpha=1, label='Correct classified Label 2')
if len(test_lin_error_1) > 0:
    ax.scatter(np.array(test_lin_error_1)[:, 0], np.array(test_lin_error_1)[:,
1], s=20, color='tab:green', alpha=1, label='Incorrect classified from Label 1
as Label 2')
if len(test_lin_error_2) > 0:
    ax.scatter(np.array(test_lin_error_2)[:, 0], np.array(test_lin_error_2)[:,
1], s=20, color='tab:red', alpha=1, label='Incorrect classified from Label 2 a
s Label 1')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of test datasets with correct and incorrect classifications')
ax.legend()
plt.show()
```



In [420]:
```
print("The accuracy is: {}".format(test_lin_score))
print("The total number of errors are: {}".format(len(test_lin_error_1) + len(
test_lin_error_2)))
print("The error probability is {}.".format((len(test_lin_error_1) + len(test_
lin_error_2))/test_N))
```

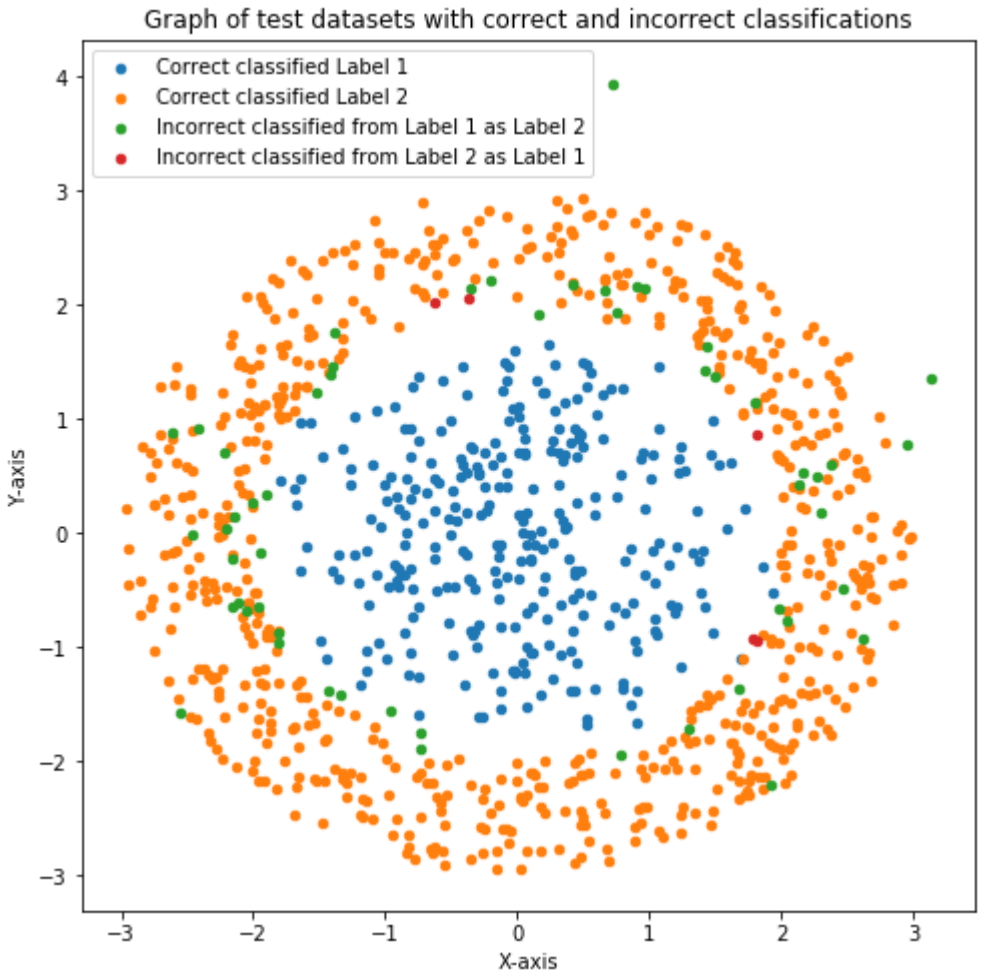
```
The accuracy is: 0.653
The total number of errors are: 347
The error probability is 0.347.
```

Gaussian-SVM

In [421]:
```
test_gaus_label = gaus_svm.predict(test_samples)
test_gaus_score = gaus_svm.score(test_samples, test_true_label)
```

In [422]:
```
test_gaus_error_1 = []
test_gaus_right_1 = []
test_gaus_error_2 = []
test_gaus_right_2 = []

for i in range(test_l_1):
    if test_gaus_label[i] == 1:
        test_gaus_error_1.append(test_samples[i])
    else:
        test_gaus_right_1.append(test_samples[i])

for i in range(test_l_2):
    if test_gaus_label[test_l_1+i] == 0:
        test_gaus_error_2.append(test_samples[test_l_1+i])
    else:
        test_gaus_right_2.append(test_samples[test_l_1+i])
```

In [423]:
```
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1)
if len(test_gaus_right_1) > 0:
    ax.scatter(np.array(test_gaus_right_1)[:, 0], np.array(test_gaus_right_1)
[:, 1], s=20, color='tab:blue', alpha=1, label='Correct classified Label 1')
if len(test_gaus_right_2) > 0:
    ax.scatter(np.array(test_gaus_right_2)[:, 0], np.array(test_gaus_right_2)
[:, 1], s=20, color='tab:orange', alpha=1, label='Correct classified Label 2')
if len(test_gaus_error_1) > 0:
    ax.scatter(np.array(test_gaus_error_1)[:, 0], np.array(test_gaus_error_1)
[:, 1], s=20, color='tab:green', alpha=1, label='Incorrect classified from Lab
el 1 as Label 2')
if len(test_gaus_error_2) > 0:
    ax.scatter(np.array(test_gaus_error_2)[:, 0], np.array(test_gaus_error_2)
[:, 1], s=20, color='tab:red', alpha=1, label='Incorrect classified from Label
2 as Label 1')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Graph of test datasets with correct and incorrect classifications')
ax.legend()
plt.show()
```



In [424]:
```
print("The accuracy is: {}".format(test_gaus_score))
print("The total number of errors are: {}".format(len(test_gaus_error_1) + len
(test_gaus_error_2)))
print("The error probability is {}.".format((len(test_gaus_error_1) + len(test
_gaus_error_2))/test_N))
```

```
The accuracy is: 0.941
The total number of errors are: 59
The error probability is 0.059.
```

Comments:

Comments on results:

We can see above that the classification for the data using the linear-SVM technique gives a very high error for a variety of parameters. On the contrary, the classification works well with the Gaussian-SVM classifier. This is mainly because the data under consideration is not linear and infact is partly gaussian. Hence, the error probability achieved by the Gaussian-SVM is very low while that for Linear-SVM is large and entirely dependent on the prior prbability of the data.

In the Linear-SVM, the linear kernel tries to classify the data in the present space which is not possible since the data is partly Gaussian. In the Gaussian-SVM, the Gaussian kernel transforms the data into a higher dimensional space to carry out its classification and is thus able to classify the data more accurately.

Support Vector Machine (SVM):

SVM or Support Vector Machine is a classifier which aims to maximise the separation distance between the boundary and the closest data points of the classes from it. But when the data is present in a non-linear form, it cannot classify the data merely on this criteria. Hence, it uses non-linear classification techniques such as the Gausian-SVM. Here, the classifier transforms the data into a higher dimension and classifies the data using a higher dimensional vector or plane trying to maximise the gap between the boundary and the closest points of the classes.

In [ ]:

In [ ]:

Note:

- The tables in the program meant as part of the graphical representation of the various values of the hyperparameters are not visible in the PDF. Hence, I have attached pictures of the same here for reference.

Linear-SVM:

Index no.	C	Accuracy
0	0.5	0.6439999999999999
1	1.5	0.6439999999999999
2	2.5	0.6439999999999999
3	3.5	0.6439999999999999
4	4.5	0.6439999999999999
5	5.5	0.6439999999999999
6	6.5	0.6439999999999999
7	7.5	0.6439999999999999
8	8.5	0.6439999999999999
9	9.5	0.6439999999999999
10	10.5	0.6439999999999999
11	11.5	0.6439999999999999
12	12.5	0.6439999999999999
13	13.5	0.6439999999999999
14	14.5	0.6439999999999999
15	15.5	0.6439999999999999

```
The optimal hyperparameter pair is C = 0.5 found at index number 0
```

Gaussian-SVM:

Index no.	C	Gamma	Accuracy
28	1.5	2.25	0.945
29	1.5	2.5	0.944
30	1.5	2.75	0.945
31	1.5	3	0.9480000000000001
32	1.5	3.25	0.9480000000000001
33	1.5	3.5	0.9490000000000001
34	1.5	3.75	0.9490000000000001
35	1.5	4	0.9490000000000001
36	1.5	4.25	0.9490000000000001
37	1.5	4.5	0.9490000000000001
38	1.5	4.75	0.95
39	1.5	5	0.95
40	2.5	0.25	0.944
41	2.5	0.5	0.942
42	2.5	0.75	0.942
43	2.5	1	0.943

```
The optimal hyperparameter pair is C = 1.5 and Gamma = 4.75 found at index number 38
```

- The codes for the above solutions can be accessed through the following GitHub link. They codes have been uploaded in both, a python file as well as a jupyter notebook.
  https://github.com/nandayvk/EECE5644_ML_HW_4.git