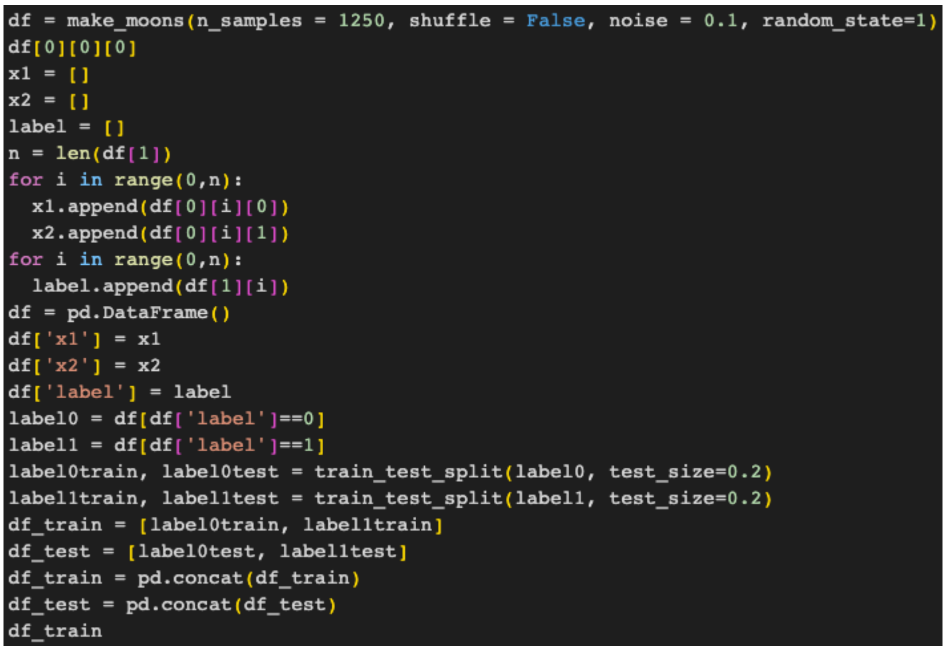
MTH 4330 Take Home Final - Ely Bardavit

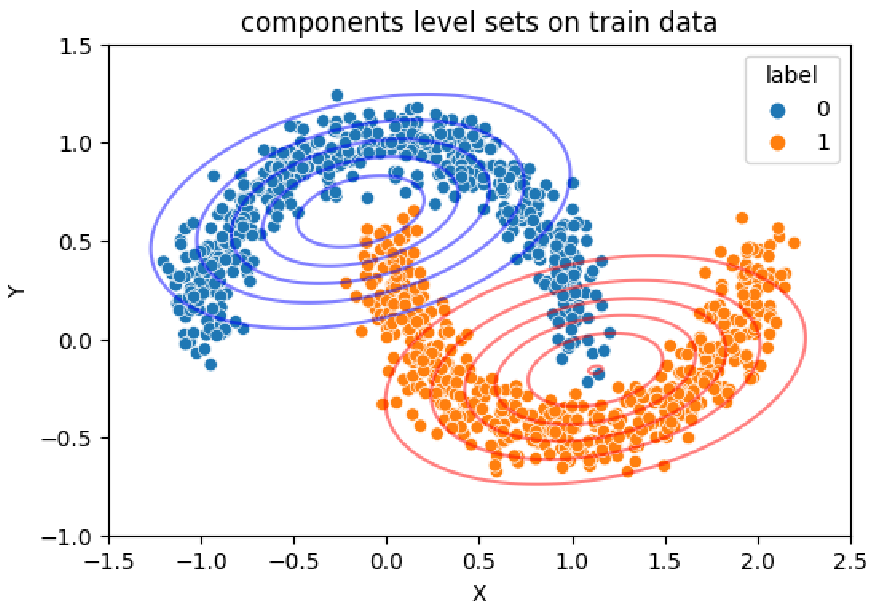
GMM as a density estimator algorithm:

There isn't a data set called two moons; however there is a function called make_moons() in the sklearn library. With that function I generated 1250 points (675 points belonging to the top moon and 675 belonging to the bottom moon) with noise = 0.1. I then split the data equally so that 125 of each moon will be used as testing data.

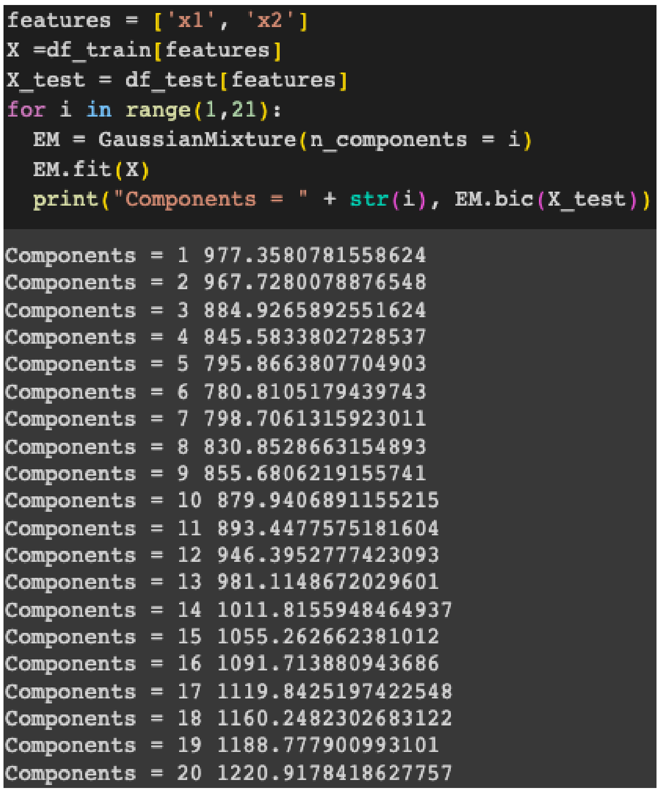
<u>Note:</u> Although the make_moons() function provides the label, they were only used to show how the gaussians fit the data in relation to their actual label.



```
df = make_moons(n_samples = 1250, shuffle = False, noise = 0.1, random_state=1)
df[0][0][0]
x1 = []
x2 = []
label = []
n = len(df[1])
for i in range(0,n):
  x1.append(df[0][i][0])
  x2.append(df[0][i][1])
for i in range(0,n):
  label.append(df[1][i])
df = pd.DataFrame()
df['x1'] = x1
df['x2'] = x2
df['label'] = label
label0 = df[df['label']==0]
label1 = df[df['label']==1]
label0train, label0test = train_test_split(label0, test_size=0.2)
label1train, label1test = train_test_split(label1, test_size=0.2)
df_train = [label0train, label1train]
df_test = [label0test, label1test]
df_train = pd.concat(df_train)
df_test = pd.concat(df_test)
df_train
```



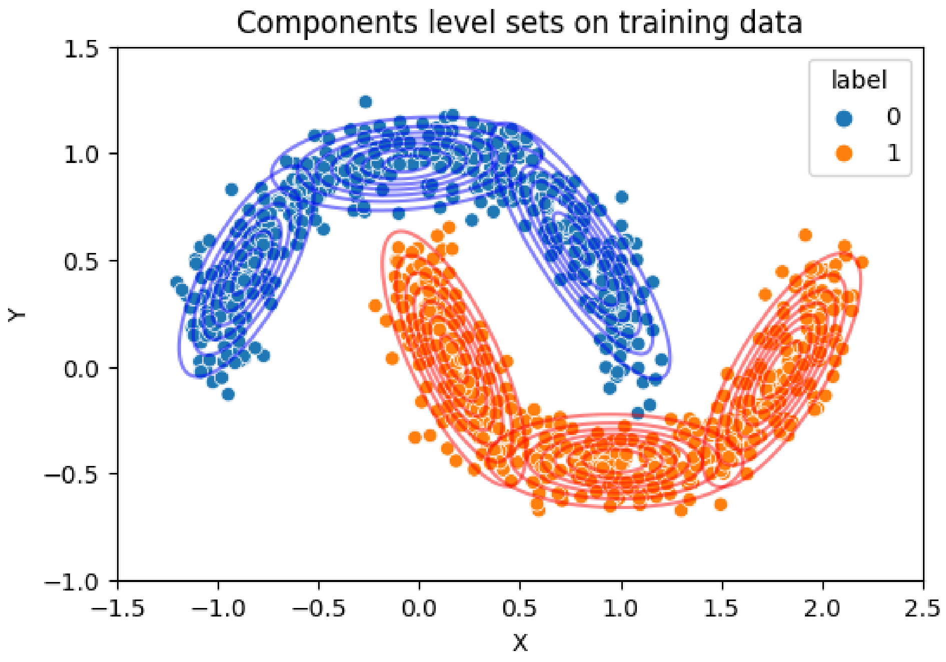
When using a two component GMM, we can see that for the plot that the model is not able to capture the data well. We are already given the labels of these moons thanks to the make_moons() function so we can see that the ends of the blue moon and the orange moon are in the centre of gaussian corresponding to the other moon. Because of this it would be wise to increase the number of components.



```
features = ['x1', 'x2']
X =df_train[features]
X_test = df_test[features]
for i in range(1,21):
  EM = GaussianMixture(n_components = i)
  EM.fit(X)
  print("Components = " + str(i), EM.bic(X_test))
```

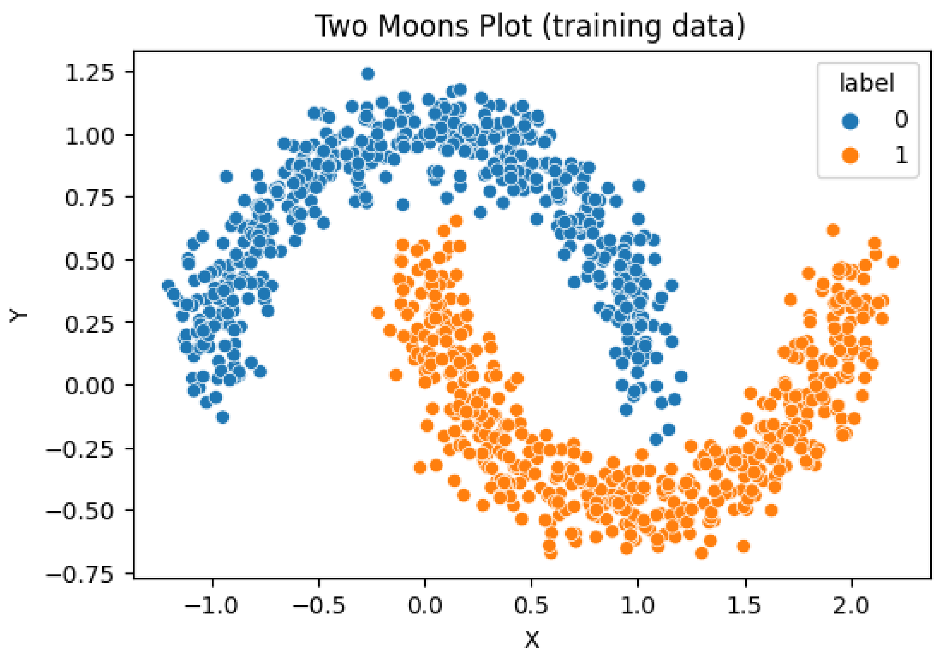
```
Components = 1 977.3580781558624
Components = 2 967.7280078876548
Components = 3 884.9265892551624
Components = 4 845.5833802728537
Components = 5 795.8663807704903
Components = 6 780.8105179439743
Components = 7 798.7061315923011
Components = 8 830.8528663154893
Components = 9 855.6806219155741
Components = 10 879.9406891155215
Components = 11 893.4477575181604
Components = 12 946.3952777423093
Components = 13 981.1148672029601
Components = 14 1011.8155948464937
Components = 15 1055.262662381012
Components = 16 1091.713880943686
Components = 17 1119.8425197422548
Components = 18 1160.2482302683122
Components = 19 1188.777900993101
Components = 20 1220.9178418627757
```

Using the BIC criterion on the test set, we discover that the optimal number of components to use is 6 because the BIC criterion achieves a minimum at 6 components.

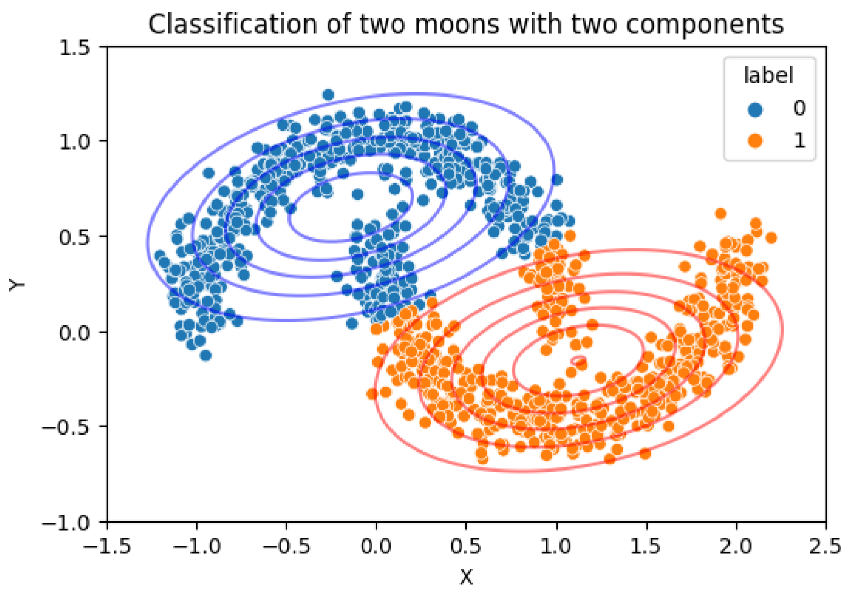


As we can see, given the labels, a 6 component gaussian mixture model fits the data well. However, it is not perfect as there are a few points on the upper moon that are in a gaussian of the bottom moon. This is most likely due to the 0.1 noise we established for our two moons. The BIC criterion tells us that if we increase the number of components because of those few points, we would be overfitting the model.

GMM as a classifier:



For classification with 2 components, we can utilise the .predict() function of the GaussianMixture library since the number of components of the model will be equal to the number of classes.



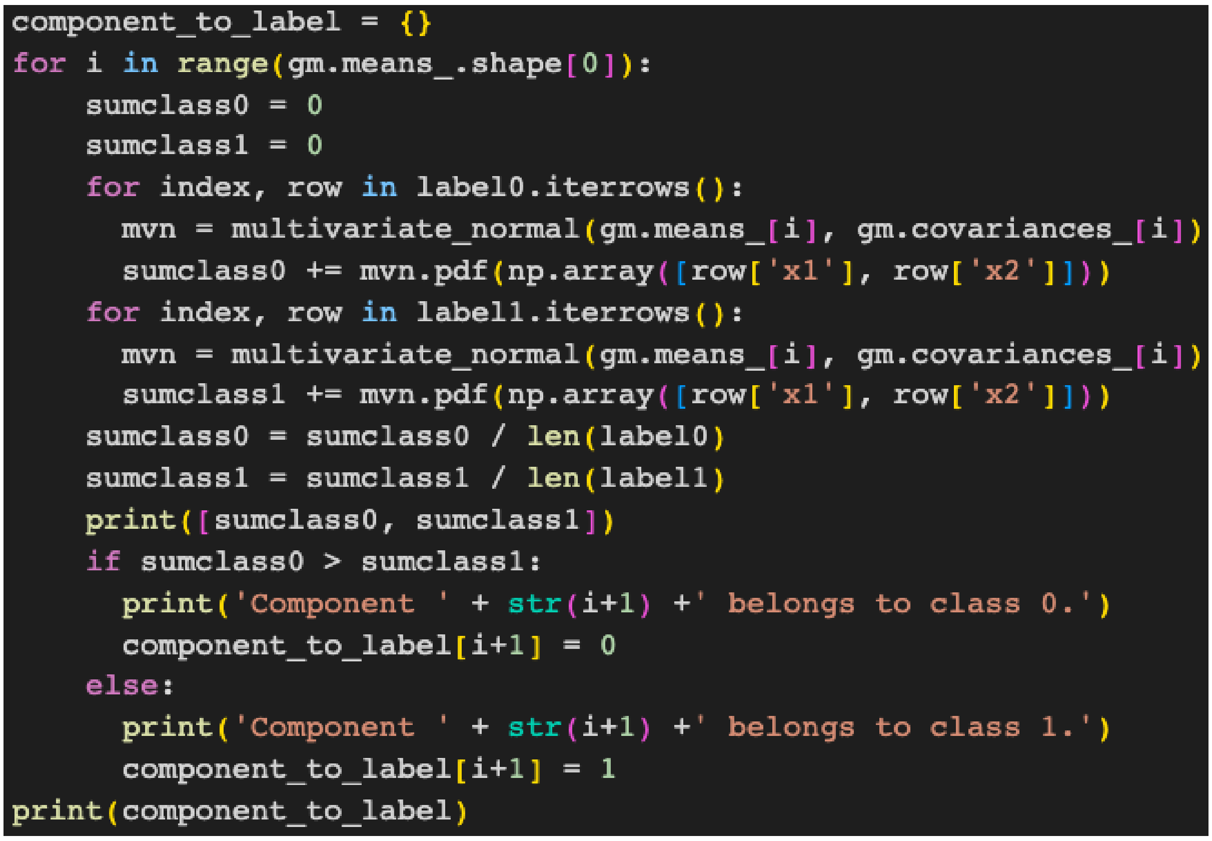
As we can see, with just two components, the gaussian mixture model does not classify the points on the ends of the moon well at all. In order to fix this, we can use the 6 component model from part 1 of the project. The problem with this is that we would have more components than classes, so we can no longer conveniently use the .predict() function.

To solve this, we have to assign each gaussian to a class. To do this, I will be using the algorithm suggested in class which is:

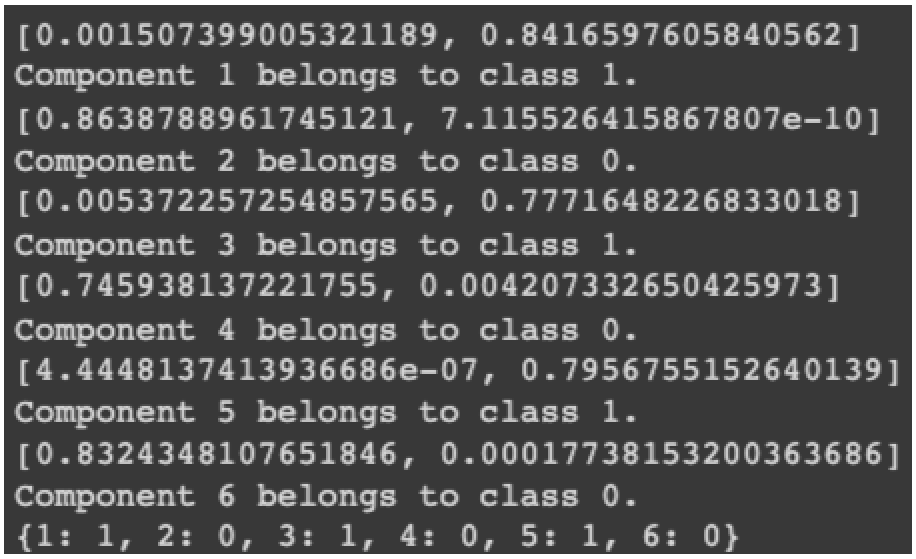
$$\frac{1}{N_{\mathrm{I}}}\sum_{i} N_k(x_i^{\mathrm{I}}|\overrightarrow{\mu_k}, \Sigma_k)$$

$$\frac{1}{N_{\mathrm{II}}}\sum_{i} N_k(x_i^{\mathrm{II}}|\overrightarrow{\mu_k}, \Sigma_k)$$

For each component, I will check which quantity is bigger and assign it to class I or class II (I will correspond to the moon with label 0 and II with label 1).

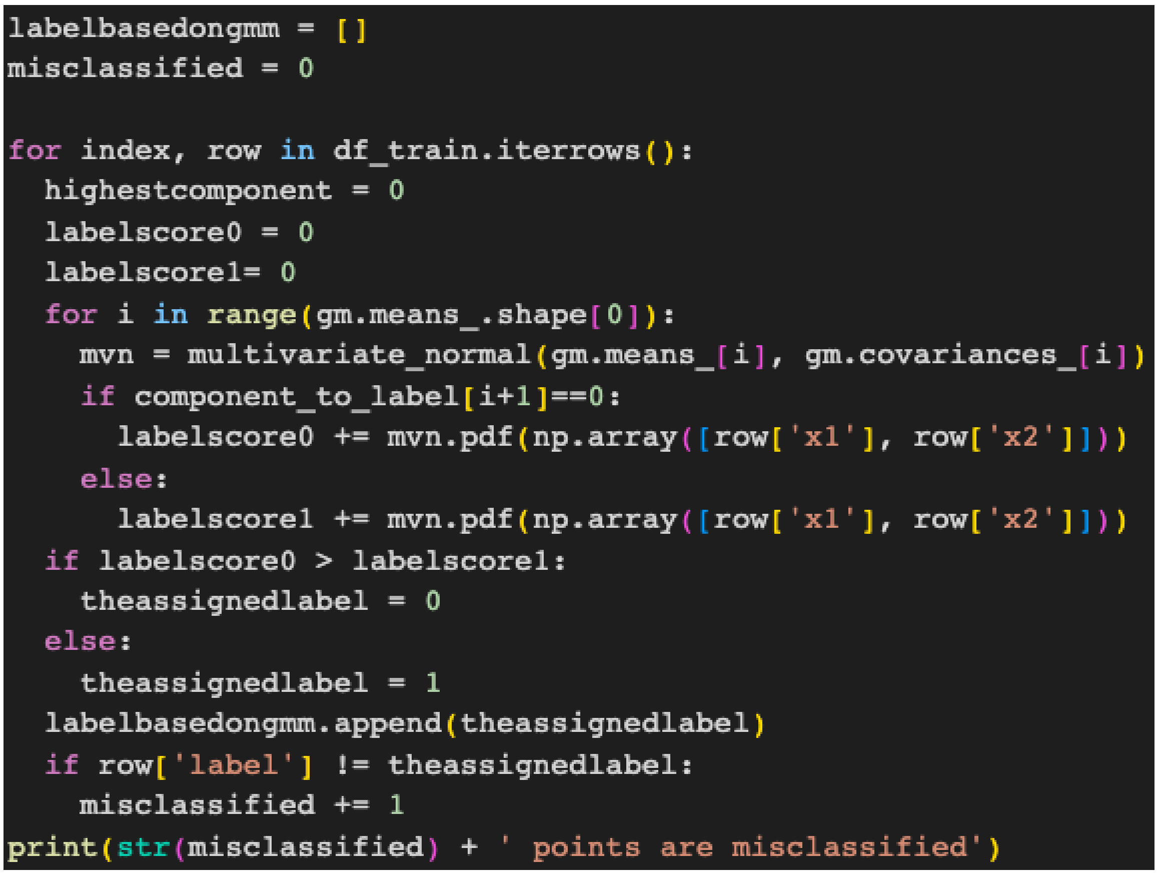


```
component_to_label = {}
for i in range(gm.means_.shape[0]):
    sumclass0 = 0
    sumclass1 = 0
    for index, row in label0.iterrows():
      mvn = multivariate_normal(gm.means_[i], gm.covariances_[i])
      sumclass0 += mvn.pdf(np.array([row['x1'], row['x2']]))
    for index, row in label1.iterrows():
      mvn = multivariate_normal(gm.means_[i], gm.covariances_[i])
      sumclass1 += mvn.pdf(np.array([row['x1'], row['x2']]))
    sumclass0 = sumclass0 / len(label0)
    sumclass1 = sumclass1 / len(label1)
    print([sumclass0, sumclass1])
    if sumclass0 > sumclass1:
      print('Component ' + str(i+1) +' belongs to class 0.')
      component_to_label[i+1] = 0
    else:
      print('Component ' + str(i+1) +' belongs to class 1.')
      component_to_label[i+1] = 1
print(component_to_label)
```



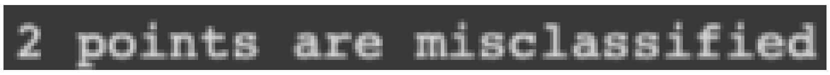
```
[0.0015073990053211889, 0.8416597605840562]
Component 1 belongs to class 1.
[0.8638788961745121, 7.115526415867807e-10]
Component 2 belongs to class 0.
[0.005372257254857565, 0.7771648226833018]
Component 3 belongs to class 1.
[0.745938137221755, 0.004207332650425973]
Component 4 belongs to class 0.
[4.444813741393668 6e-07, 0.795675515264013 9]
Component 5 belongs to class 1.
[0.832434810765184 6, 0.00017738153200363686]
Component 6 belongs to class 0.
{1: 1, 2: 0, 3: 1, 4: 0, 5: 1, 6: 0}
```

This is the algorithm I wrote in order to classify the components based on the label that the component will assign. Next we must classify each point in the training data.



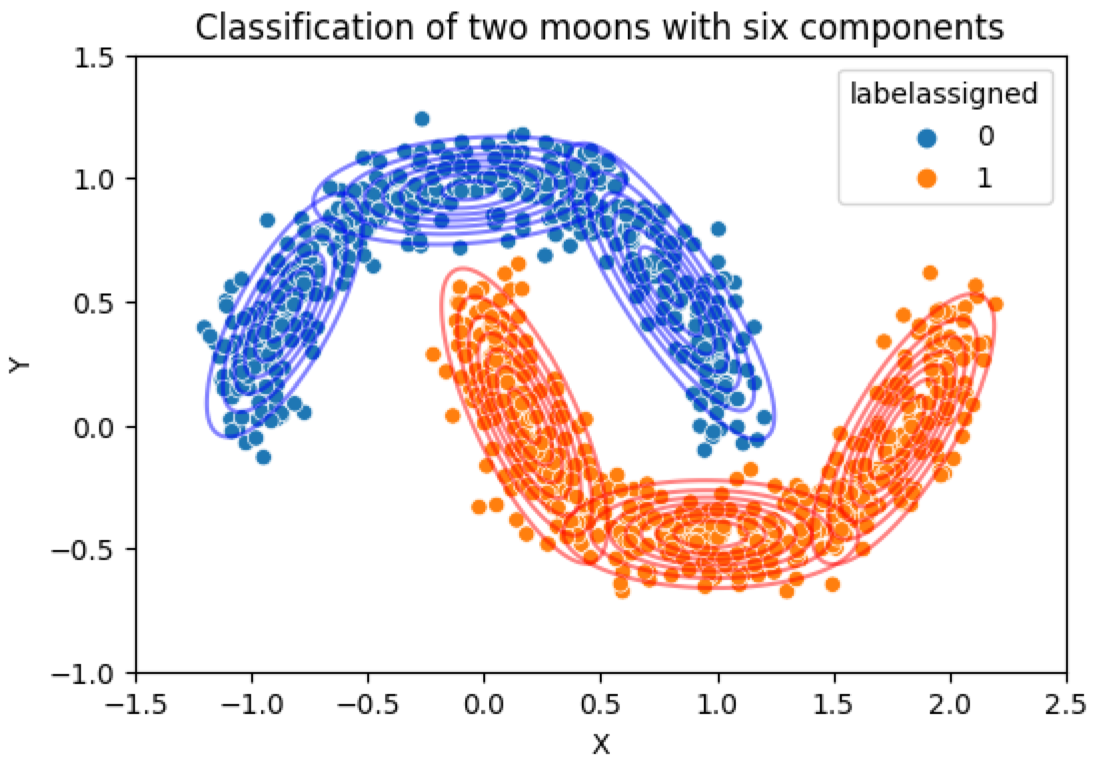
```
labelbasedongmm = []
misclassified = 0

for index, row in df_train.iterrows():
  highestcomponent = 0
  labelscore0 = 0
  labelscore1= 0
  for i in range(gm.means_.shape[0]):
    mvn = multivariate_normal(gm.means_[i], gm.covariances_[i])
    if component_to_label[i+1]==0:
      labelscore0 += mvn.pdf(np.array([row['x1'], row['x2']]))
    else:
      labelscore1 += mvn.pdf(np.array([row['x1'], row['x2']]))
  if labelscore0 > labelscore1:
    theassignedlabel = 0
  else:
    theassignedlabel = 1
  labelbasedongmm.append(theassignedlabel)
  if row['label'] != theassignedlabel:
    misclassified += 1
print(str(misclassified) + ' points are misclassified')
```

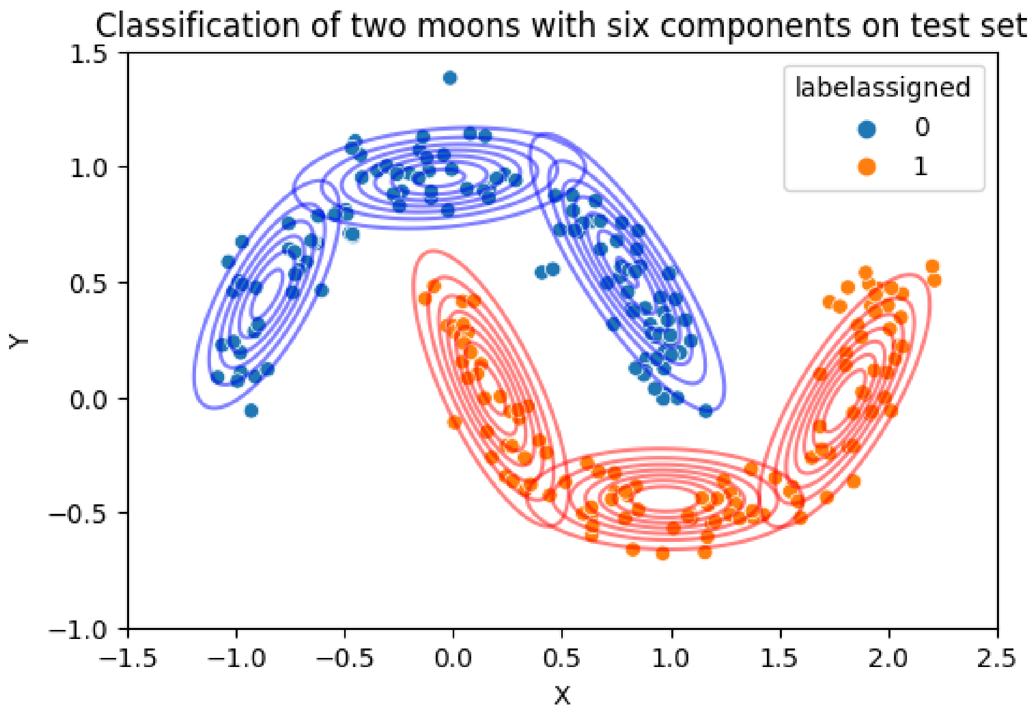


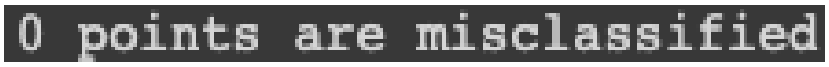
```
2 points are misclassified
```

This is the algorithm I wrote to classify the training data. The idea of this algorithm is to calculate the cumulative sum of the gaussians with label 0 at a point and compare it to the cumulative sum of the gaussians with label 1 at the same point. The sum which is larger will determine which label the point will be assigned to. It also prints out how many points were misclassified which out of 1000 points was only 2 points.



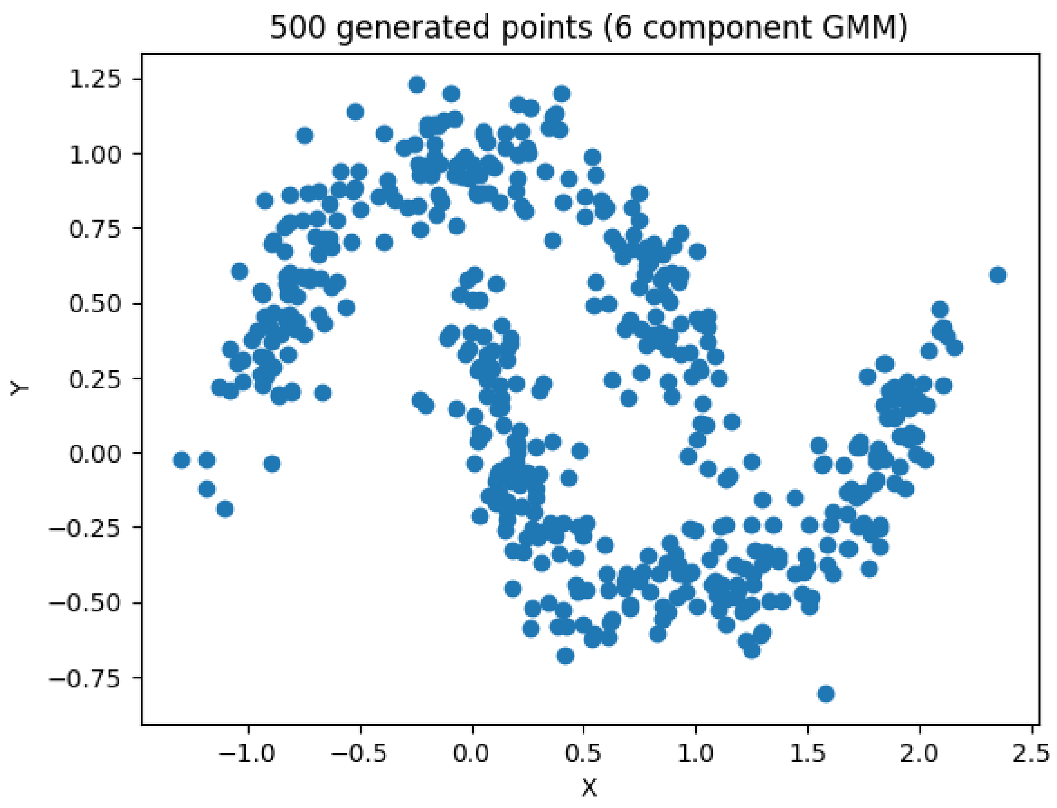
What is even better is that this model has 0 misclassified points on the test set which is 250 points, so 0 classification error. There is no need to increase the number of components needed because the model is 99.8% accurate on the training set and 100% accurate on the test set.





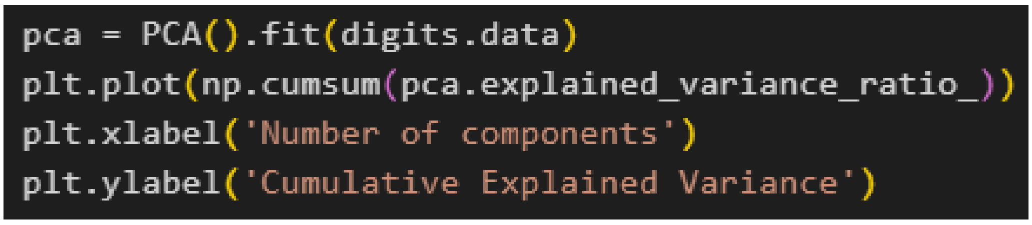
```
0 points are misclassified
```

The difference between training our gaussian mixture model with looking at the label instead of treating it like a density estimator is that we can see misclassifications and understand how to improve our model. If we create 500 samples of the 6 component gaussian mixture model, we can see the shape of two moons just as expected:

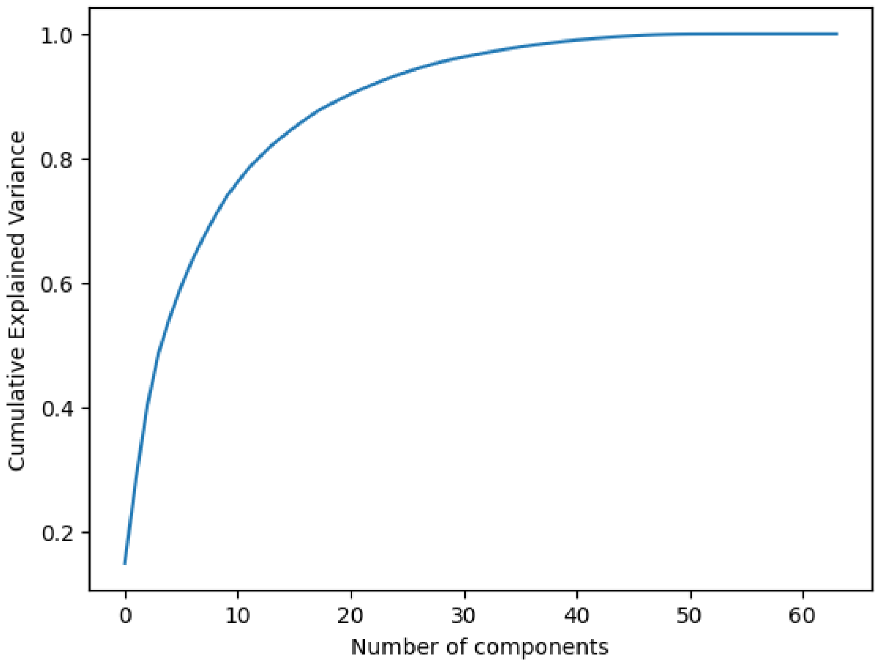


GMM on the MNIST dataset as a generative model:

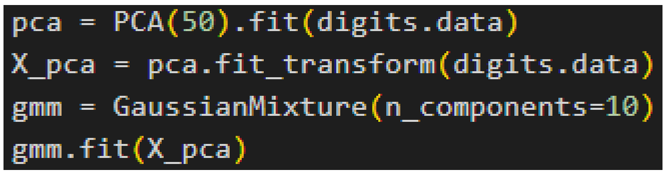
For the final part of the project, we will be using the MNIST data set provided by sklearn which can be called by the function load_digits(). The data set comes with 1797 data points with 64 components each.



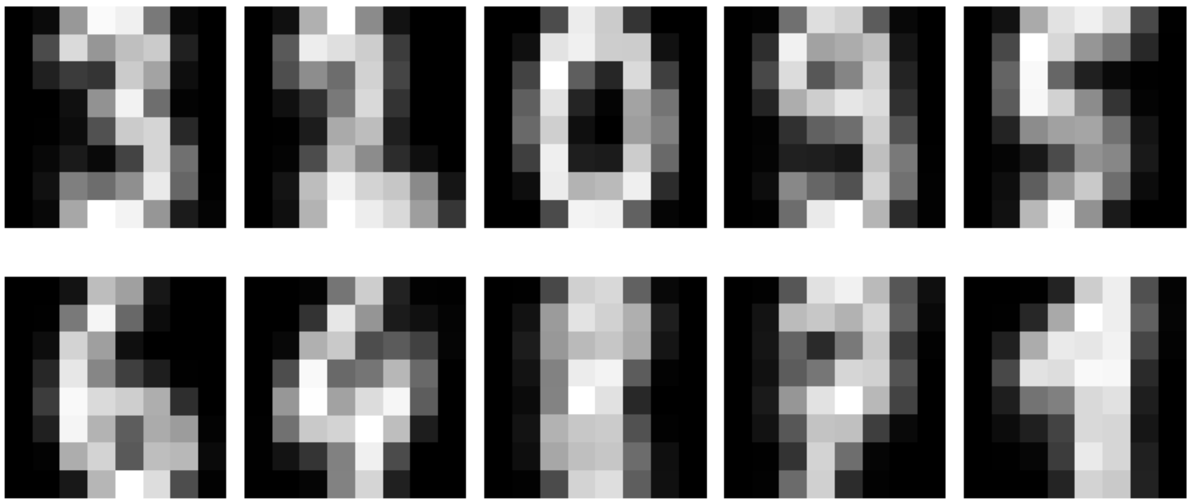
```
pca = PCA().fit(digits.data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative Explained Variance')
```



I will use the number of components to be 50 because at around 50 components nearly all variance is explained. Using 10 components for the gaussian mixture (1 gaussian for each class, I plotted the means of the gaussian mixture model as follows:

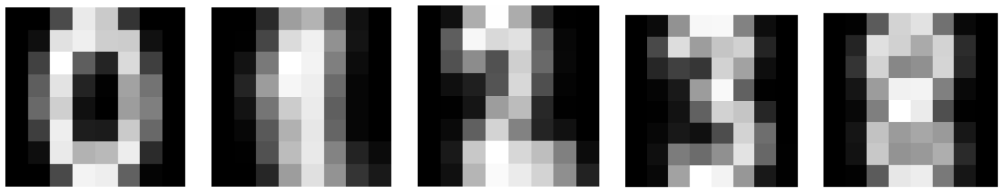


```
pca = PCA(50).fit(digits.data)
X_pca = pca.fit_transform(digits.data)
gmm = GaussianMixture(n_components=10)
gmm.fit(X_pca)
```



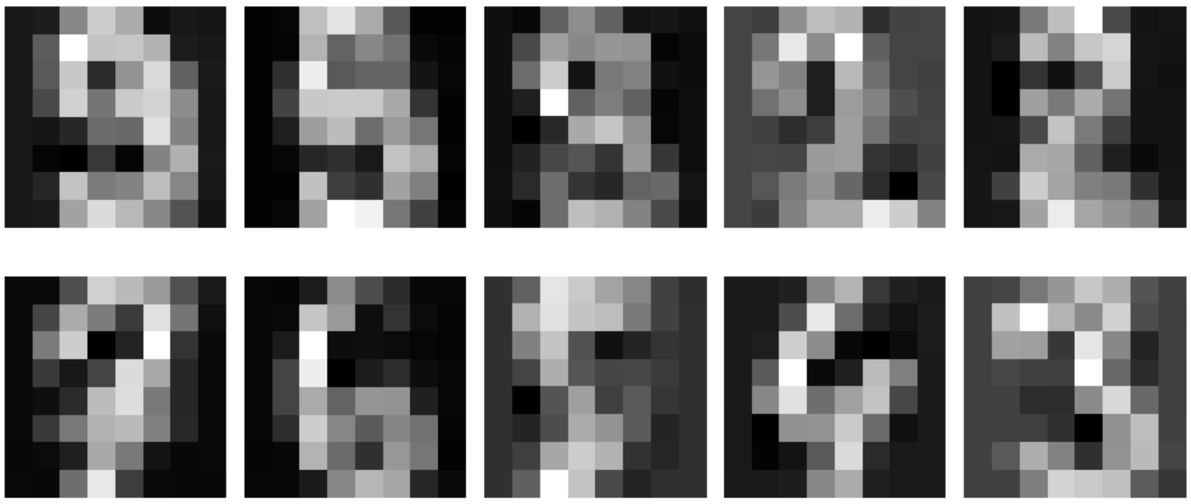
Although not so clear, all of the digits can be identified. The means are as follows: 3, 2, 0, 9, 5, 6, 4, 8, 7 and 1. This was for creating a model without looking at the labels. Increasing the number of components does not improve the clarity of the digits. It seems that the most problematic digits are 2 and 8, which is most probably due to a large variance in how they are written. One can tell it’s an 8 because the pixels get darker in the top and bottom middle of the 8.

My next idea is to divide the MNIST data set into labels and train a gaussian model on each one of the separated data sets.



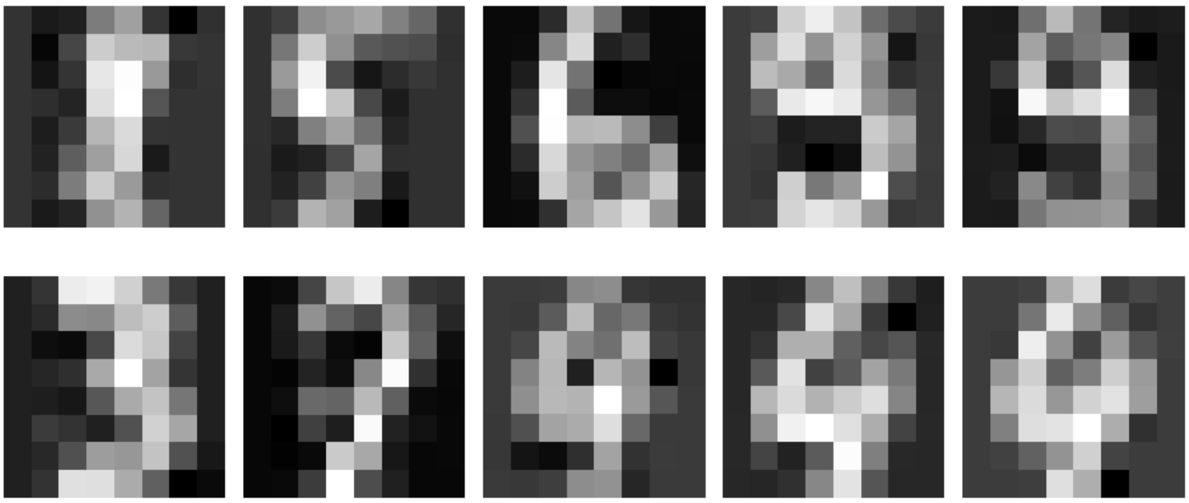
However, after doing this for 5 digits , I had realised that these were producing nearly identical means as when I was not looking at the labels, as shown above. My final idea was to use the separated labelled data sets to and put a 10 component gaussian on each one of the data sets. I would then select the best component from each model and concatenate the means, covariances and weights into a combined list. However, this did not work as it did not produce readable means. The reason that I think this did not work is because the weights that I had used for the combined and concatenated gaussian mixture model were relative to the gaussian mixture models that I had extracted from. Therefore, because the weights of the components were not accurate, it was not a good model.

With that being said, I decided that the best model is the one that was created when the labels were not shown and had 10 components. Generating sample points looks as follows:

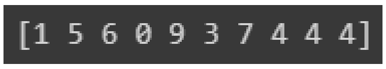


This data for the most part looks like handwritten digits. The digits are 9, 5, 9, 2, 2, 7, 6, 5, 4 and 3.

To check if the gaussian mixture model is performing well as a classifier, I created a boosted gradient tree using the digits reduced to 50 components, as stated earlier, and the label. I then created sample points from the GMM and made the boosted gradient tree classify those sample points. Here is an example:

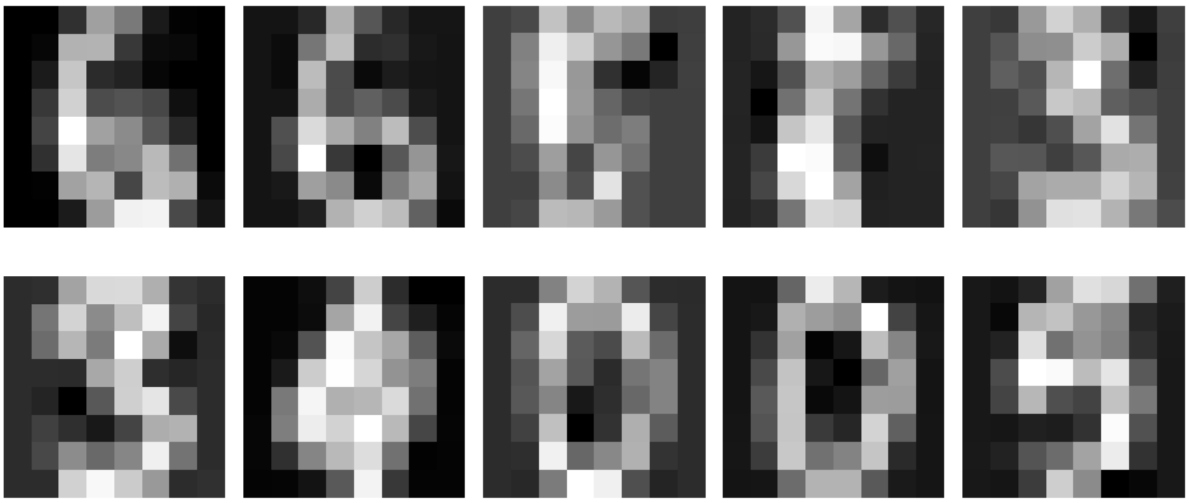


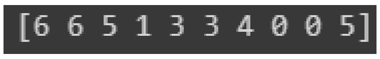
The boosted gradient tree labelled these samples as:



```
[1 5 6 0 9 3 7 4 4 4]
```

Here is another example:





```
[6 6 5 1 3 3 4 0 0 5]
```

In conclusion, since the gaussian mixture model with 10 components is able to generate sample points/digits and the boosted gradient tree is able to classify most of the points correctly, I would say that this gaussian mixture model is performing well. It is also interesting to see how such a simple algorithm like gaussian mixture can be utilised to generate handwritten digits when it is conventionally suggested to use a generative adversarial network instead. This is not to say that this gaussian mixture model is perfect but it gets the job done.

Link to python notebook:
https://colab.research.google.com/drive/1LTVSHXNkmPMTedOtB-Dr6fETF9E4Jw8h?usp=sharing