MTH 4330 - Take Home Final

Jake Li

## GMM as a density estimator algorithm

#### 1) Download the data set relative to two clusters of points displayed as two moons.

I generated 3000 points with a noise of 0.1 from sklearn.datasets.make\_moons and plotted the points.

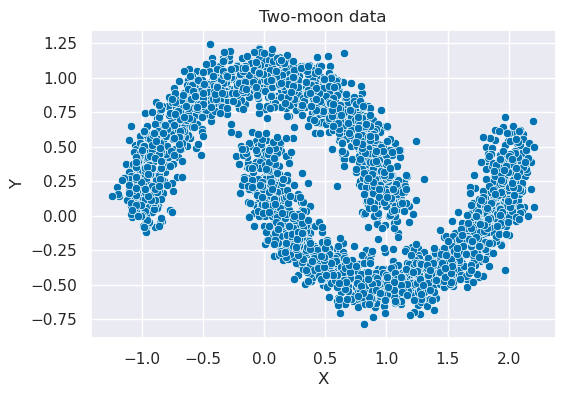


Fig 1. Scatter plot of the two-moon dataset generated from scikit-learn.

#### 2) Build a two component GMM and plot the result visualizing the data and the level sets of the two Gaussians (plot of two different colors the level sets relative to the two different Gaussians).

First, I fitted all the data to the Gaussian Mixture model with 2 components. Then, to plot the level sets of the components, I need to plot the PDF of the multivariate normal distribution with the mean and covariance given by each component. The following code shows how I plotted the contour lines.

# get coordinates

k = gm.means\_.shape[0]

x = np.linspace(-1.5, 2.5)

y = np.linspace(-1.0, 1.5)

x\_grid, y\_grid = np.meshgrid(x, y)

coordinates = np.array([x\_grid.ravel(), y\_grid.ravel()]).T

# plot contour lines

fig, ax = plt.subplots(figsize=(6,4))

sns.scatterplot(data=df, x='x', y='y', hue='label', ax=ax).set(title='Gaussians level sets', xlabel='X', ylabel='Y')

colors = ['orange', 'blue']

for i in range(k):

mean = gm.means\_[i]

cov = gm.covariances\_[i]

z\_grid = multivariate\_normal(mean, cov).pdf(coordinates).reshape(x\_grid.shape)

ax.contour(x\_grid, y\_grid, z\_grid, colors=colors[i], alpha=0.5)

Result:

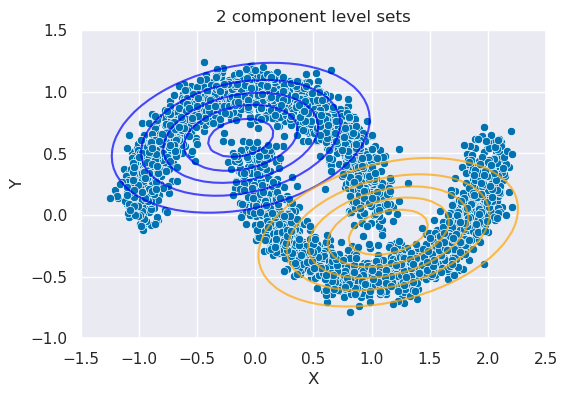
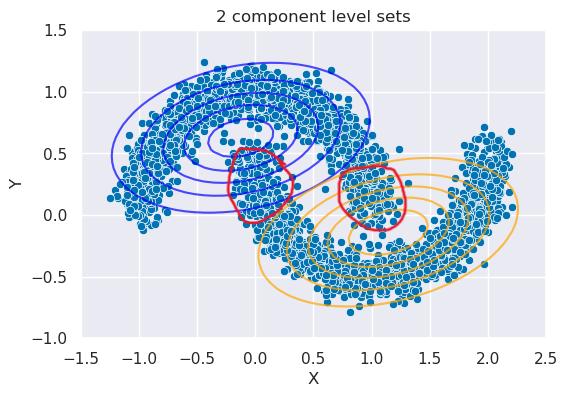


Fig 2. Level sets of the two mixture components from GMM.

#### 3) What do you observe? If we were to use Gaussian mixture as a clustering algorithm (each point is assigned to the component that has the largest value when evaluated at that point), would you say that a two components GMM is able to capture well the two clusters?

If using the Gaussian mixture as a clustering algorithm, it would perform badly because a lot of points will be misclassified. In the following plot, the points inside the red circles will be misclassified. This is because the points inside the red circle are closer to the wrong Gaussian. For example, the points in the red circle on the left will be classified as the upper moon when they should be classified as the lower moon.

Fig 3. Modified Fig 2.

#### 4) If you are thinking that a solution is to increase the number of components, how would you choose such number?

First, I split the data into training and testing sets because I want to see how well the model performs on the test set. The following plot shows the points from the training and testing set after stratified sampling.

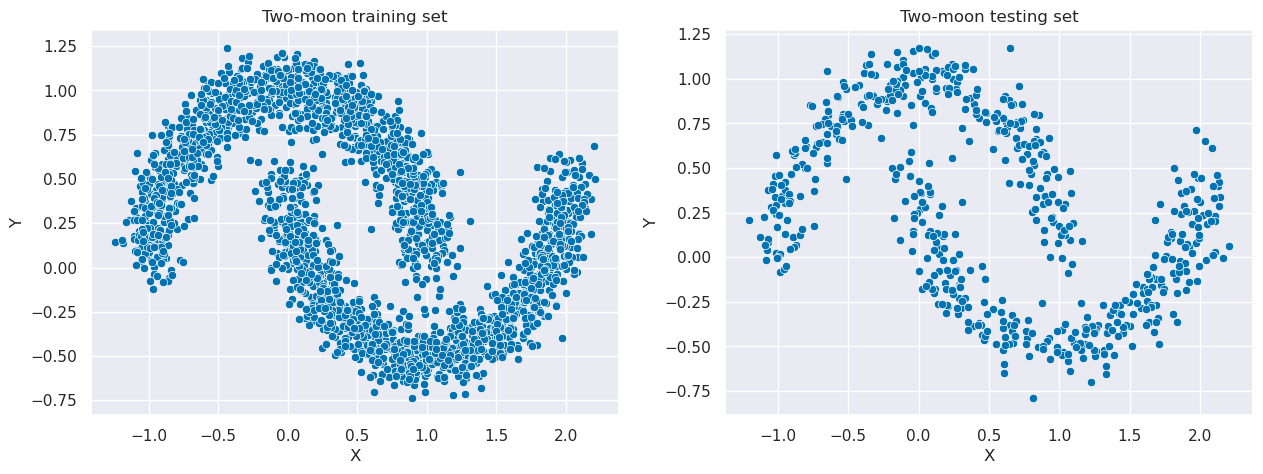


Fig 4. Scatter plot of the training and testing set after stratified sampling.

To find the optimal number of components, I used the GridSearchCV function and the BIC score as the criterion which penalizes models with more parameters. The GridSearchCV function will find the best hyperparameters by finding the lowest BIC score out of all the possible combinations of models in the parameter grid I define:

params = {

'n\_components': range(2,16),

'covariance\_type': ['full', 'tied']

}

The parameter grid I defined above will test the model with components from 2 to 15 and a covariance type of “full” (each component has its own general covariance matrix) or “tied” (all components share the same general covariance matrix). For example, it will calculate the BIC score for all the following models:

GaussianMixture(n\_components=2, 'covariance\_type=‘full’)

GaussianMixture(n\_components=2, 'covariance\_type=’tied’)

GaussianMixture(n\_components=3, 'covariance\_type=‘full’)

…

GaussianMixture(n\_components=15, 'covariance\_type=‘full’)

GaussianMixture(n\_components=15, 'covariance\_type=‘tied’)

The best parameters GridSearchCV found with the lowest BIC score is using 6 components and a covariance type of “full”. Here is the BIC score bar plot that shows the BIC scores for all the 28 possible models.

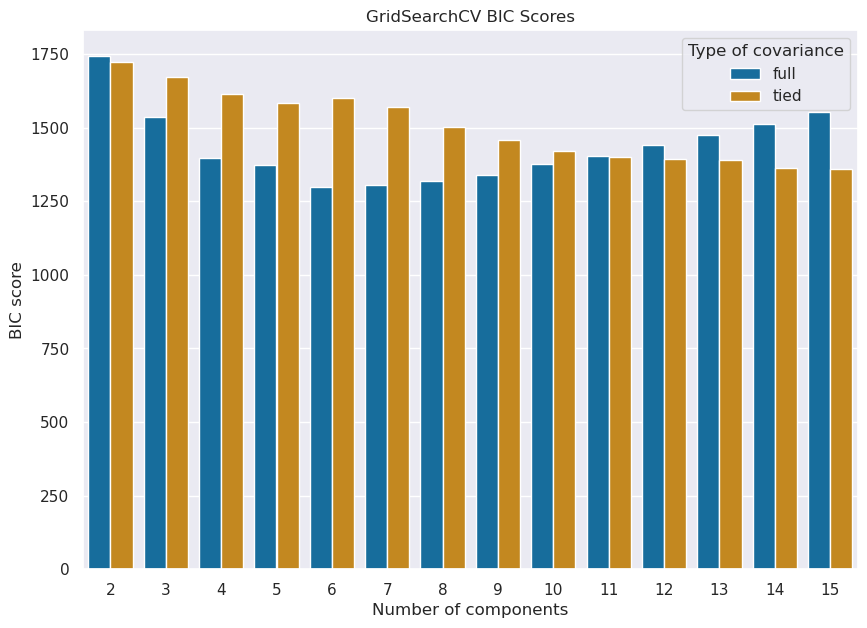


Fig 5. BIC score vs the number of components by type of covariance.

I created a GMM with the best parameters found and trained it on the training set. Then I plotted the component level sets on both the training and testing set.

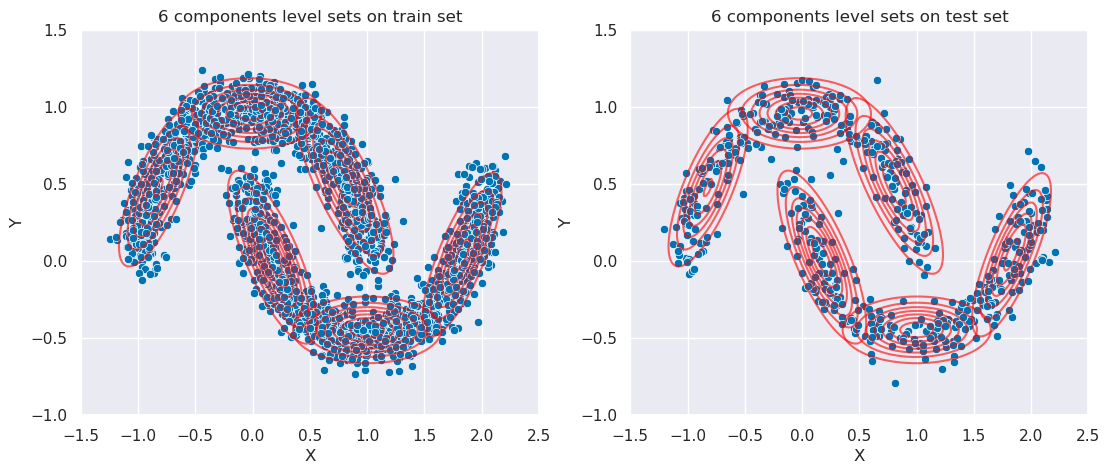


Fig 6. Level sets of the GMM with 6 components and “full” covariance on training and testing set.

These 6 components captured the data really well and I can see why it has the lowest BIC score.

## GMM as a classifier.

#### 1) Plot the data.

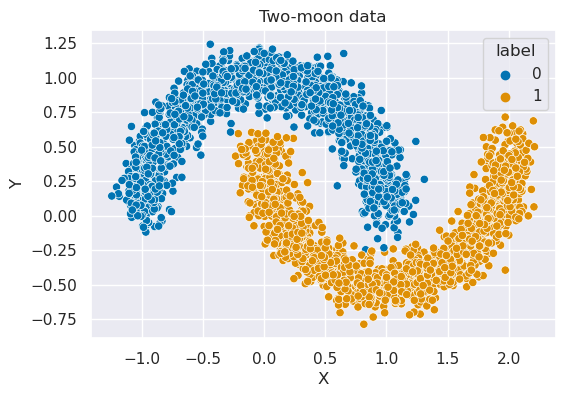


Fig 7. Scatter plot of the two-moon dataset generated from scikit-learn with labels.

#### 2) You can use a GMM to perform classification using two components, one for each class. After you trained your GMM each point can be assigned to a class by looking at P(X|Y = i) where i can be ±1.

I split the data into training and testing sets, trained the 2 components GMM on the training set, got the prediction for the testing set, and then plot the prediction results with the labels on the testing set:

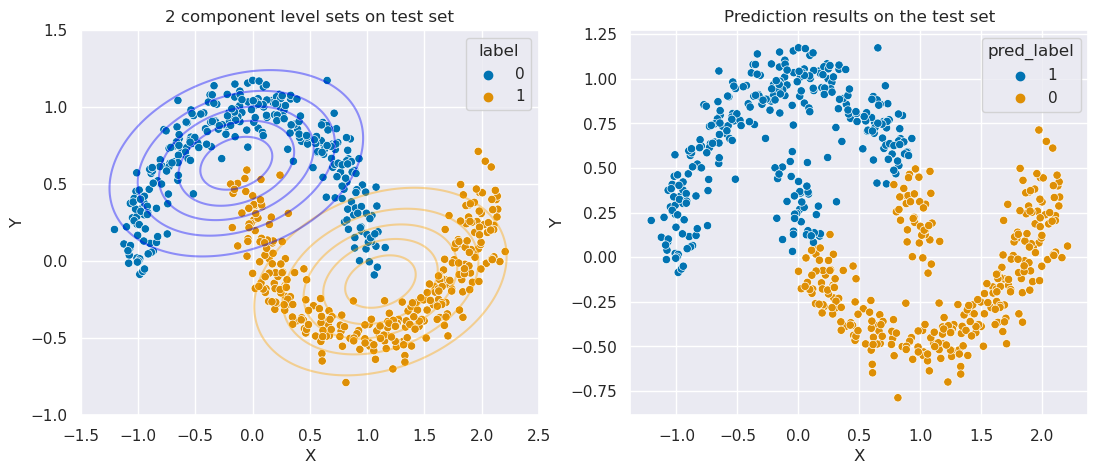


Fig 8. Comparison between testing sets with original labels and predicted labels.

#### 3) How can we use your GMM to perform classification when you use more components than classes?

First, I assigned each component to a label by evaluating the mean of the multivariate normal probability density at each component for both labels, 0 and 1. For example, if the probability density of component 1 for label 0 is higher than the probability density of component 1 for label 1, then component 1 will be assigned to label 0.

After I assigned each component to a label, I used the model to predict which component each point in the test set belonged to. Then the point is assigned to the label that was assigned to the component. For example, if point 1 is predicted to be in component 3 and component 3 is assigned to label 0, then point 1 is assigned to label 0.

#### 4) Plot the points together with the level sets of the GMM components and color the points according to what class is the GMM assigning them. What do you observe?

The following code shows how I tested out the idea from above (question 3) starting with 4 components:

# GMM with 4 components fitted with training set

gm = GaussianMixture(n\_components=4, covariance\_type='full', random\_state=0).fit(X\_train)

# Assign a label to each component as described above

label0 = train\_data[train\_data['label']==0]

label1 = train\_data[train\_data['label']==1]

component\_labels = []

for i in range(gm.means\_.shape[0]):

score0 = np.mean(multivariate\_normal.pdf(label0[['x', 'y']], gm.means\_[i], gm.covariances\_[i]))

score1 = np.mean(multivariate\_normal.pdf(label1[['x', 'y']], gm.means\_[i], gm.covariances\_[i]))

if score0 > score1:

component\_labels.append(0)

else:

component\_labels.append(1)

# Get predictions of the test set (model predicts which component each point belongs to)

y\_pred = gm.predict(X\_test)

# Change the predictions to a label of 0 or 1

y\_pred\_label = []

for x in y\_pred:

y\_pred\_label.append(component\_labels[x])

Then I plotted the results of using the model as a classifier to assign labels.

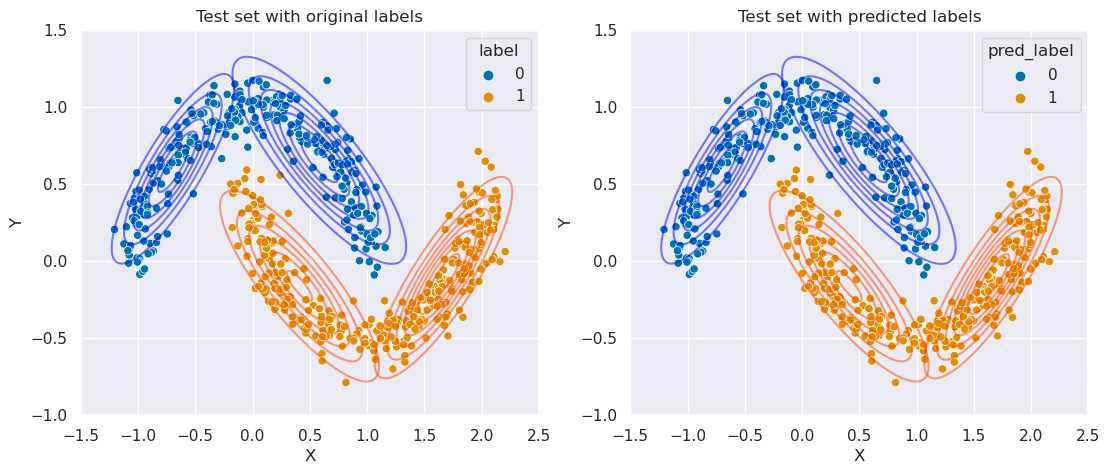


Fig 9. GMM with 4 components as a classification - Comparison between original labels and predicted labels on the test set with component level sets.

From Fig 9, you can see that only one point from the test set was misclassified. I also calculated the accuracy score and I got 0.9983.

#### 5) If needed you can increase the number of components of your GMM by minimizing the classification error.

GMM with 4 components already has an accuracy of 0.9983, I wonder if more components will have a perfect prediction. Therefore, I built a for loop that gets the different model accuracies on the testing set after training the model on the training set. I tested the number of components from 2 to 15 and the covariance types of “full” and “tied”. Here is the result:

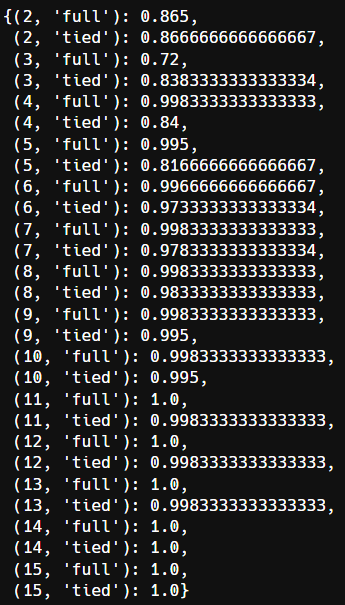


Fig 10. Model accuracy per # of components and covariance type.

Although we can get a perfect prediction with 11 components using “full” covariance, I think GMM with 4 components is better since we use fewer components and 11 components feels like overfitting.

#### 6) What is the difference with training your GMM without looking at the label and just considering it as a density estimator algorithm?

I don’t think there’s much difference because the process of getting the Guassians is the same with or without the labels. In fact, the GMM in scikit-learn ignores the label and only takes into account the points. I would say the difference is that GMM as a density estimator algorithm is unsupervised learning and GMM as a classifier is supervised learning. GMM as a classifier is very interesting because it actually works very well on classification even though it’s an unsupervised learning method.

#### 6) Generate 500 points from your GMM, does it seem to you they are reproducing the two moons?

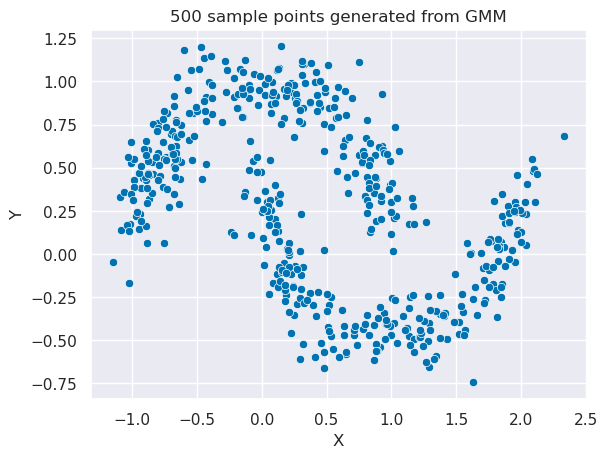


Fig 11. 500 points generated from the GMM with 6 components and covariance type of “full”.

From Fig 11, we can clearly see that the 500 points generated from GMM closely resemble the two moons dataset.

## GMM on the MNIST dataset as a generative model.

#### 1) Perform PCA on the original dataset. Justify the number of components you are choosing.

First, I loaded the MNIST dataset. Then, for the number of components in PCA, I decided to use a number of components that will explain at least 95% of the variance. The following code shows how I found the number of components:

pca = PCA(n\_components=0.95).fit(digits.data)

pca.n\_components\_

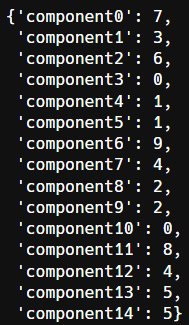
I put n\_components=0.95 to tell the model to select the number of components such that the amount of variance that needs to be explained is greater than 95%. Then I ran pca.n\_components\_ to see the number of components which is 29. Finally, I applied dimensionality reduction to the MNIST dataset.

#### 2) Build a GMM as you did in part one of this final. Train first a GMM without looking at the labels (considering it as a density estimator) and then using the labels. Choose what you believe is the best in between the two models you obtain.

The best estimator I found for GMM as a density estimator using GridSearchCV with BIC criterion only has 8 components with a covariance type of “tied”. The best estimator I found for GMM as a classifier has 15 components. I chose to use the GMM as a classifier because it’s easier to evaluate model performance.

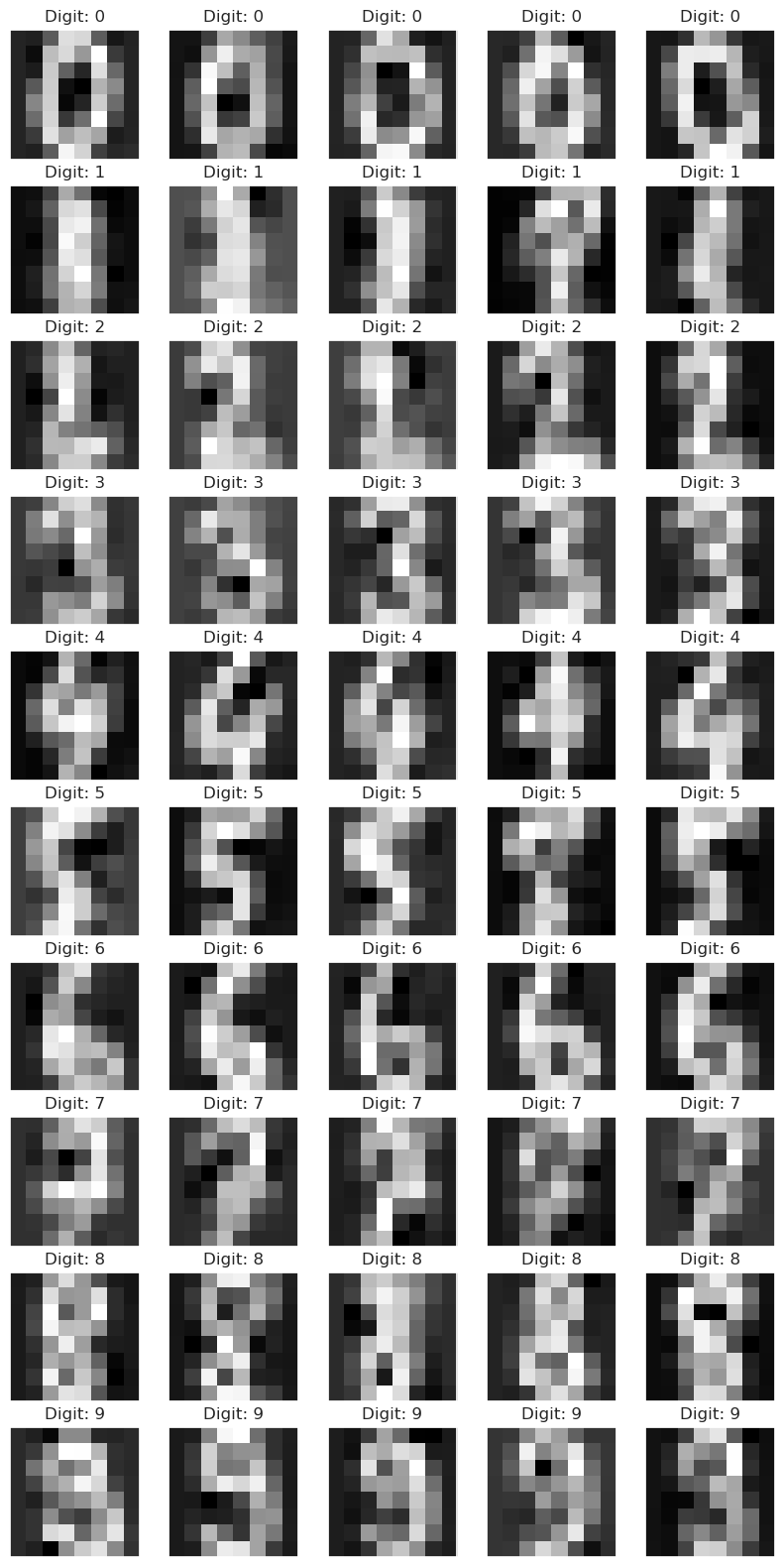
#### 3) For each class of the dataset (that is characterized by 10 classes each one representing a digit), find the GMM components belonging to that class and plot their means. If you did things correctly, each mean should be identifiable as a digit.

Here, I applied the same idea from the GMM as a classifier section question 3 where I assigned each component to a label (digit in this case). The following is the result of assigning each of the 15 components to a label (digit):

Fig 12. Component labels.

#### 4) Sample the GMM and plot some of these samples, do they look like handwritten digits?

I sampled 100 points from the GMM and then I transformed the data back to the original space with 64 features using the inverse\_transform method from the PCA model. I chose 5 random samples from each digit and plotted them:

Fig 13. Generated digits.

Based on Fig 13, I would say they look very alike to handwritten digits, just a bit blurry.

#### 5) Another way to check whether your GMM is performing well as a classifier is to compare its performance by using a different classifier, like, for instance, a boosted gradient tree. If you generate points from the GMM and assign them the label based on which component the point is coming from (remember that each component is assigned to a given class), does your tree classify that point correctly?

First, I fitted a random forest classifier on the original and normalized MNIST dataset. I used 80% of the data to train the model and got an accuracy of 0.977 on the test set. The model performed very well and so I moved on to test the model with the data that’s generated by GMM.

I generated 1000 samples from GMM and transformed the data back to the original space with 64 features using the inverse\_transform method from the PCA model. Then I predicted the labels using the random forest classifier and got an accuracy of 0.941.

Since the model performed very well on the data generated by GMM, I can conclude that the GMM I created is performing well as a classifier.