Homework 4

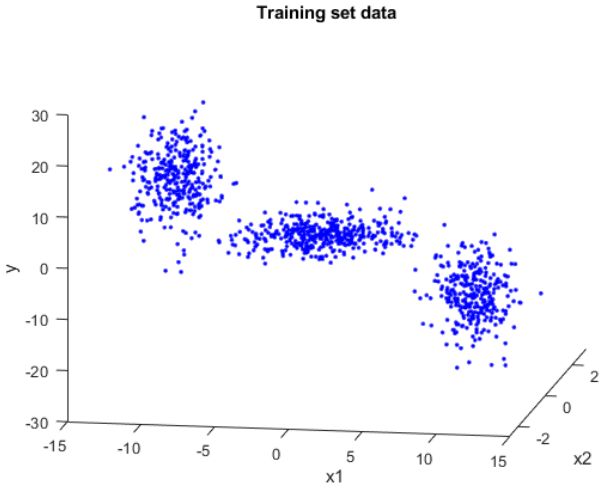
Daniel Potapov

Question 1

Data Distribution

For this question, I reused the dataset generation code from Assignment 2, which creates a Gaussian mixture distribution that resembles an upwards S curve. It would then be the job of the neural network to recognize this behavior and tune its parameters during training properly.

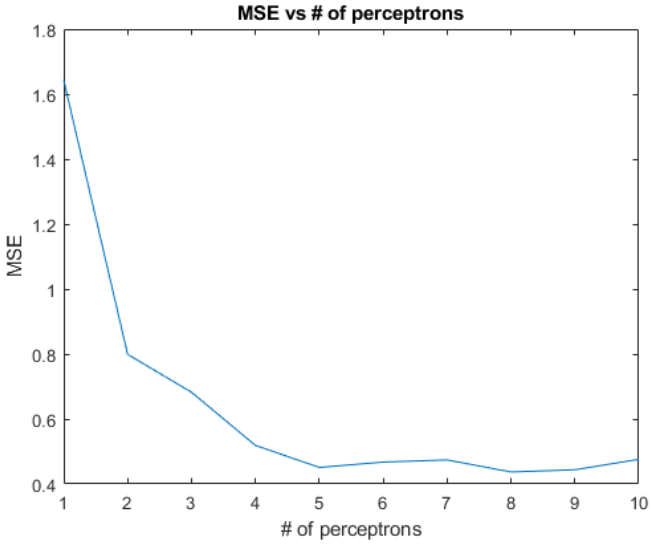
Here is a visual representation of the data distribution using the training set:



Cross Validation

For the training set, there could be multiple orders of models applied to it, and each of these selections can perform differently. Therefore, we performed 10-fold cross validation to find the best perceptron count for each data set.

Much like in the previous assignment, it seemed as if cross validation over a number of perceptrons could not identify a local minimum, which didn't increase my confidence in the model selection, but the performance of the network later on was on par with expectations.



Neural Network Packages and Verification

During cross validation and training, I opted to use the *feedforwardnet* network object that comes with the MATLAB's Deep Learning Toolbox. I had to do some simple refactoring from my use of *patternnet* in the previous assignment, which I had verified using the GUIs that the Deep Learning Toolbox provided. For this assignment, I decided to perform the same litmus tests.

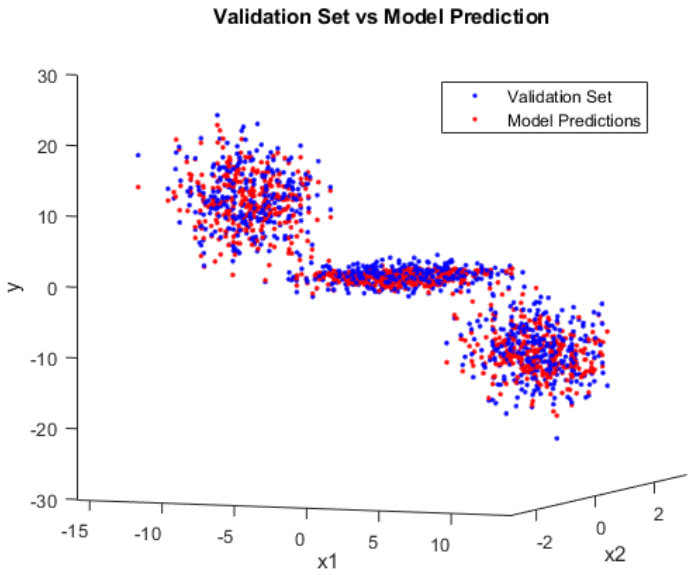
During cross validation, I wanted the only variable between tests to be the number of perceptrons used and the subset of the data used to train the network. To do this, I turned off initial randomization of network weights and biases by setting MATLAB's random number generation to the default seed of 0. To ensure that my networks behaved the way I wanted them to, I took advantage of the convenient GUIs that the Deep Learning Toolbox provided, which showed me the weights, biases, and overall structure of the network. As cross validation and training ran, I manually inspected the network using this GUI until I was confident that the tool was providing me the functionality I expected.

Model Training

With the appropriate number of perceptrons selected (with the k-fold cross validation run shown above, this turned out to be 8 perceptrons), a neural network was trained using the entire training set. Since there could be local optima that the network could get stuck in during optimization, I trained the network with different initial weights and biases 10 times and chose the best performing network among them. Indeed, for multiple runs of training a network with the same training set, the Mean Squared Error (MSE) for each network could vary considerably, meaning many of these networks were getting trapped in local optima (by defining a sufficiently large maximum iteration count, I was confident that network training was not stopped prematurely by this count).

Performance Assessment

For each of the iterations described in the model training above, I assessed the performance of the neural network by running it on the 100 thousand sample validation set and calculating the MSE for each network. With the best network chosen out of the 10 iterations I ran the validation set on it one more time to visually compare the network's predictions against the set's actual outputs:



The MSE calculated for the validation prediction was 3.68, which is quite reasonable given the range of values for the data distribution. Visually, we can see that the network models the distribution quite well and does not suffer from the slight warping on the edges seen in the model fit from assignment 2.

Question 2

Data Distribution

In this problem, data was obtained by reading in pixel data from two images and formatting those images as feature vectors.

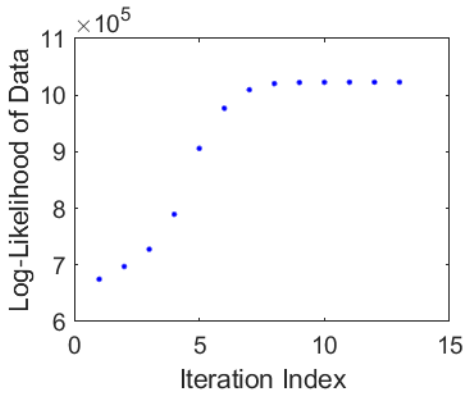
Each feature would have five components, two for the indices of the pixel and three for its RGB values. All vectors were then scaled to fit within the unit hypersphere of the vector space.

2-component Clustering

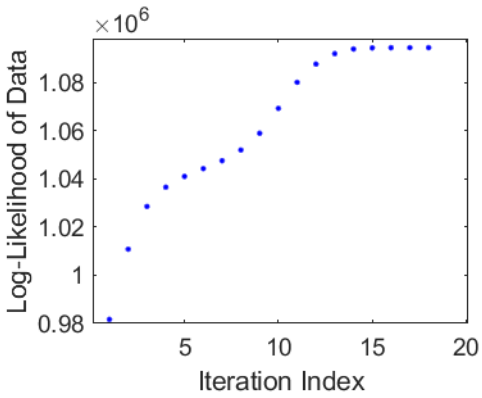
In this question we are concerned with GMM-based clustering, where clusters are determined based on what component of the Gaussian mixture some sample belongs to. Therefore, choosing the number of components in the mixture will have a significant impact on the result of the clustering. In this part, we pre-selected the number of components to be 2. In the next section, we will calculate the BIC scores for many Gaussian Mixture Models and select a component number based on those scores.

While training the model, I wanted to monitor the progress of convergence, so I opted to adapt some code that was demonstrated in lecture instead of using a MATLAB function. This came at the cost of missing some more detailed information about the distribution that objects like *gmdistribution* provides, however all I needed for this question was to know the parameters of each component and that components proportion (alpha value).

Likelihood Convergence for Plane Image



Likelihood Convergence for Bird Image



The trained 2-component GMM did not offer much on its own; each sample needed to be evaluated by each component of the mixture individually and decided into 1 of 2 classes, one for each component. Given the resemblance to classification problems discussed early in the semester, it seemed obvious to choose a rule like MAP to classify each sample into clusters. This rule would consider the alpha value of each component, which would be important if one component was considerably smaller than another.

Here is the result of clustering using a 2-component GMM and MAP classification:

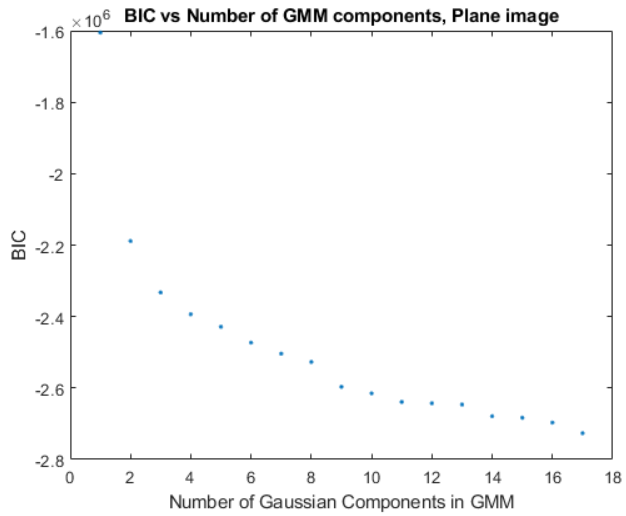
Looking at it qualitatively, I feel like the clusters do a good job of segregating the image into meaningful segments. The plane image is separated into a plane cluster and a sky cluster, and the bird image is separated into a foreground cluster and a background cluster.

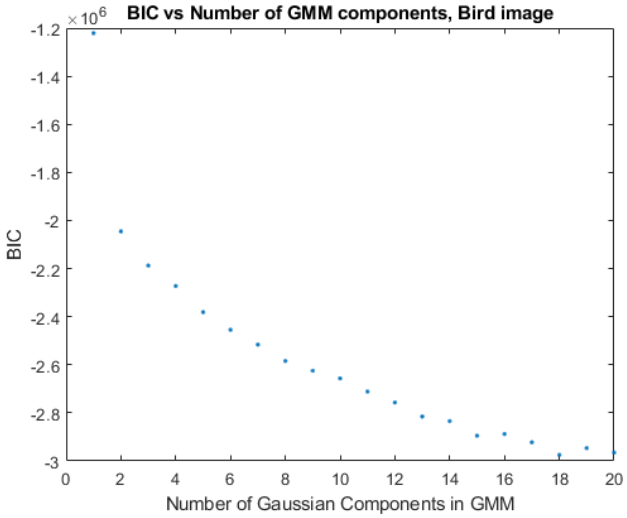
BIC and Order Selection

Like with k-fold validation in the question above, an algorithmic approach to model order selection could provide valuable insights and create a model that fits the data better. In this case, we used Bayesian Information Criterion (BIC), a quantitative measure of how good the model is at fitting the given data. The objective function used for this problem was:

$$BIC(M) = -2 \ln p_M\,(x;\ \Theta) + n \ln(dS)$$

Where the first term is the negative log likelihood of the samples using the trained model and the second term is a model complexity penalty. In theory, a model's BIC score will favor models that increase the probability of a sample without relying on too many degrees of freedom in the model's parameters. For each image, 20 GMM's were trained using MATLAB's *fitgmdist* function, each with a different number of Gaussian components. Here are the BIC scores for each component:

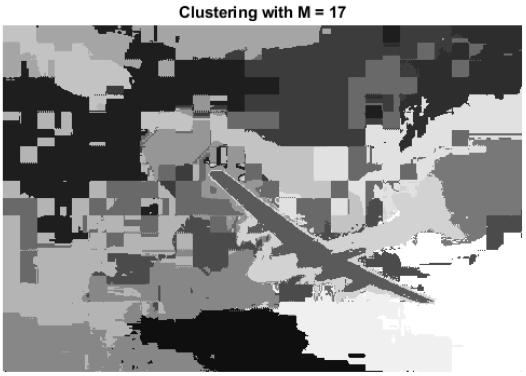




In both cases, the BIC score decreased as the number of Gaussian components in the model increased, with no local optimum found.

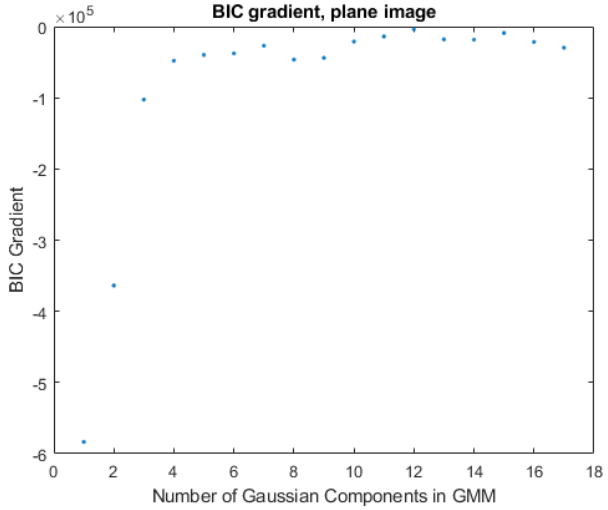
How to select order

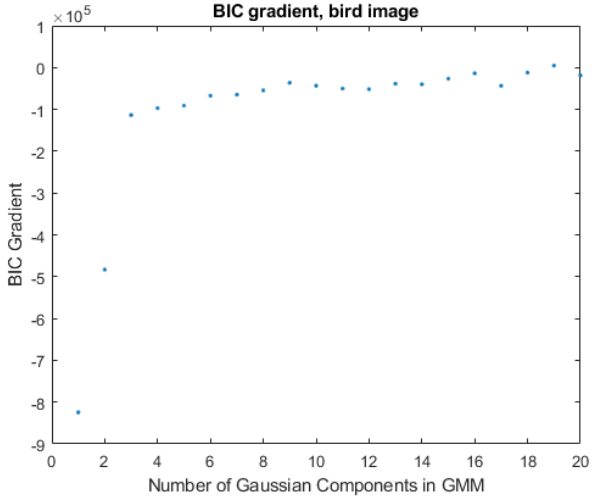
First, I chose the model order for both images based on the model that produced the lowest BIC score, although I was skeptical that this would provide a good solution. If that is the way we choose to interpret the graphs above, then I could claim that as a rule, a more complex GMM will perform better than a less complex one. This shouldn't hold up as the number of components continues to increase, since the model will suffer from overfitting the data.



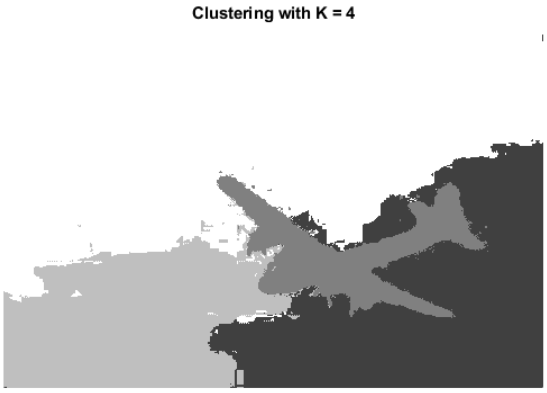
As you can see, especially in the plane image, the clustering is not very meaningful, even though the BIC score indicates that this should be the superior model for clustering. I decided to look into the intermediate values of the BIC function, and found that while the first term was on the order of -10^6, the penalty term never exceeded 1000, and grew slowly compared to the first. Therefore, it seems like by itself the BIC score does not do enough to consider overfitting of complex models.

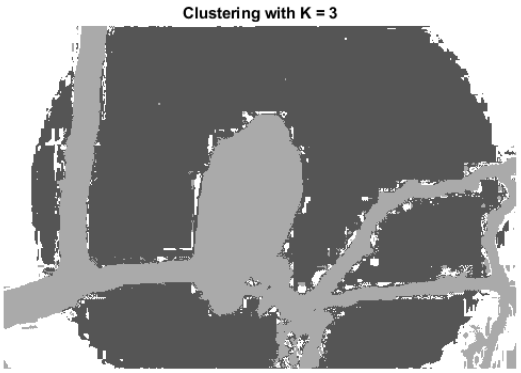
The BIC curves above may still provide useful information about model order selection however. Rather than looking at the BIC value, we could choose model order based on the BIC curve's derivative. By determining where the greatest change in performance is, we can pick a model order that provides the best improvement.





For the plane image, the gradient of the BIC curve levels out at 4 components, and the same happens with the bird's BIC curve at 3 components. This means that these model orders will provide the best improvement compared to the other orders and choosing these models with this selection rule may mean that the results will be much less prone to overfitting.





As you can see, these models performed much better at segregating different qualities of the images into their own clusters. The plane image now has separate clusters for different clouds, and the bird image now has a new cluster for the vignette effect applied to the image.