ENEE436 Project Two

Yinjun Zhang

Task One: Support Vector Machines

Intro to Support Vector Machines

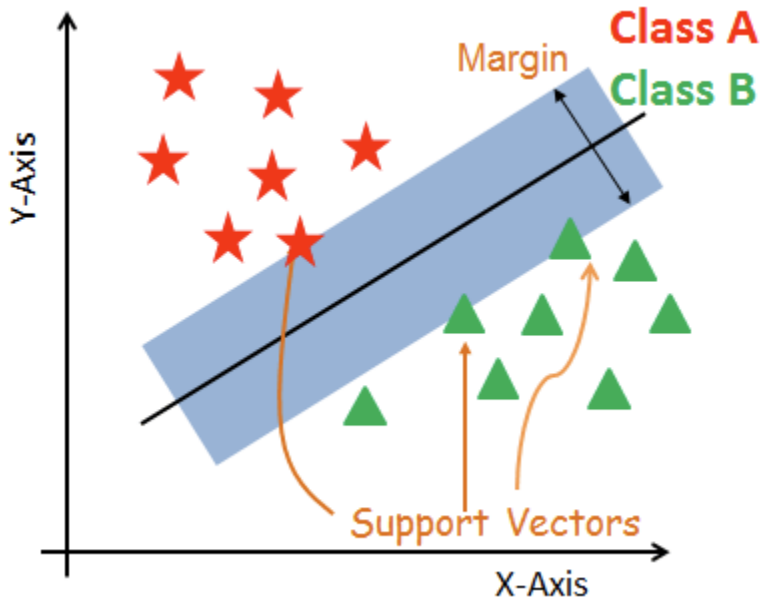


Figure 1.1: Visualization of a SVM.

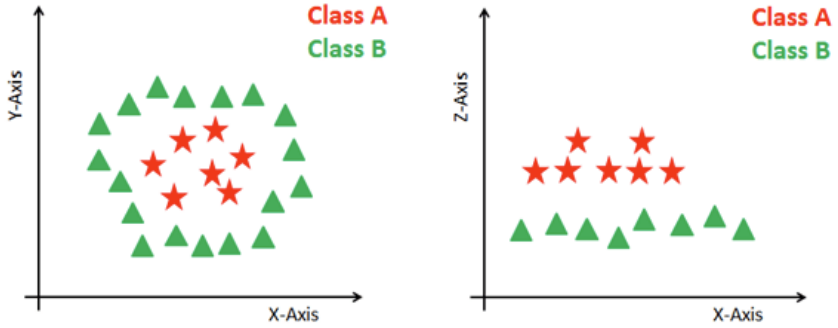


Figure 1.2: Kernel Trick Converting Non-Linear Data to Linear Data

Gaussian or Radial Basis Function (RBF)	$K(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$
Linear	$K(x_1, x_2) = x_1^\top x_2$

Figure 1.3: Linear and RBF Kernels.

Support Vector Machine (SVM) is a classification technique that works by splitting multidimensional data with a hyperplane that creates the largest possible margin between points

of different classes (Figure 1.1). Then, new points can simply be classified by deciding which side of the hyperplane they fall on. In certain cases, the original dataset cannot be split by a hyperplane as it is not linear. In such cases, we can pass the data through a kernel function, transforming it into another dataset that is linearly separable (Figure 1.2). For this project, we compare the results between a linear kernel and an RBF kernel (Figure 1.3).

Another important concept for this task is cross-validation. K-fold cross-validation takes the training data and splits it into k unique folds. Then, one of these folds is selected to be the test data while the model is trained on the other folds. Using k-fold cross-validation, we can help tune and optimize parameters of our model. In this case, we are trying to tune the sigma parameter for the RBF kernel function (Figure 1.3).

Finally, we leverage Principal Component Analysis (PCA) in this task. PCA is a reduction method that reduces the dimensionality of a dataset while preserving as much of the original meaningful data as possible. PCA is helpful for reducing super large datasets, allowing us to run calculations on them much faster without losing too much information. In this task, we will compare the accuracy of the SVM model trained on the original dataset vs. a PCA-reduced dataset.

Step-By-Step Implementation

For this task, I created one python file for each dataset we need to process: banana_svm.py, twonorm_svm.py, and waveform_svm.py. For each dataset, we first need to import all the necessary libraries. For this task, I leverage sklearn's SVM classifier so I don't have to implement one from scratch. Then, we need to initialize all the training and test datasets. Then, I run the linear SVM on the dataset and print the results. After that, I run the RBF SVM for each value of sigma stated in the task description, printing the training, test, and cross-validation results.

For the banana dataset, after deciding the optimal value for sigma, I plot the linear and RBF SVM classifications for analysis. For the waveform dataset, I run all of the above steps again on a 2-dimensional PCA-reduced dataset to analyze its effects on accuracy.

Banana Dataset

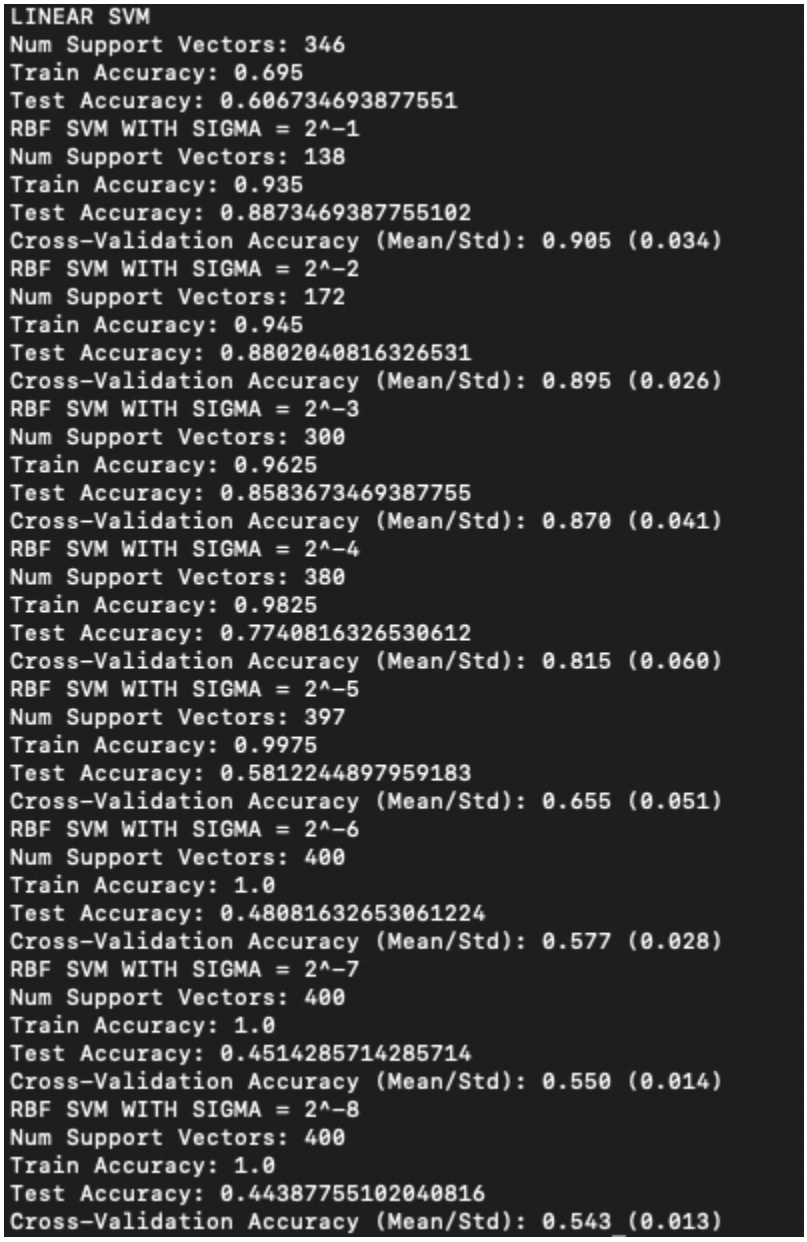


Figure 1.4: Banana Dataset Output.

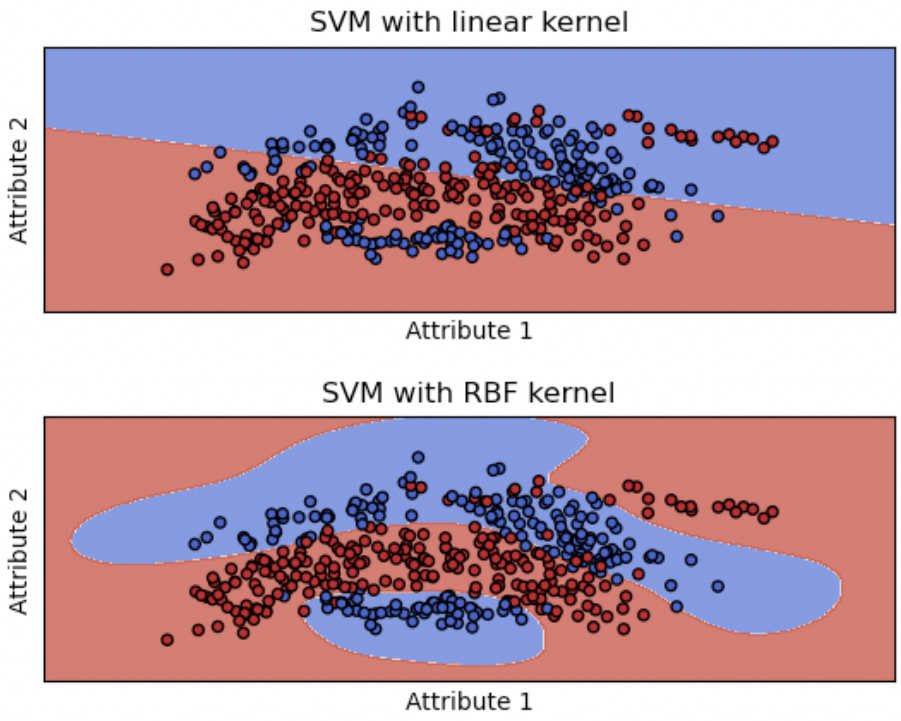


Figure 1.5: Banana Dataset Linear vs. RBF Kernel.

Looking at the output for the banana dataset (Figure 1.4), we can see that the linear SVM does decently with a training accuracy of 0.7 and a test accuracy of 0.6. However, the RBF SVM can do really well or really poorly depending on the value of sigma. For sigma = 2^-3, we get a training accuracy of 0.96 and a test accuracy of 0.86. However, when sigma = 2^-8, training accuracy goes up to 1.0, but test accuracy drops to 0.44, indicating vast overfitting. As such, the optimal sigma parameter for the banana dataset is 2^-3.

Furthermore, looking at the plots of the linear SVM vs. the optimized RBF SVM (Figure 1.5), we can visualize the accuracy of each model. The linear SVM shows a lot of points on the wrong side of the line while the RBF SVM shows that most points are within the boundaries of their side, highlighting how much more accurate an RBF SVM is with the proper parameters.

Twonorm Dataset

```
LINEAR SVM
Num Support Vectors: 33
Train Accuracy: 0.9875
Test Accuracy: 0.9667142857142857
RBF SVM WITH SIGMA = 2^-1
Num Support Vectors: 400
Train Accuracy: 1.0
Test Accuracy: 0.5148571428571429
Cross-Validation Accuracy (Mean/Std): 0.535 (0.012)
RBF SVM WITH SIGMA = 2^-2
Num Support Vectors: 398
Train Accuracy: 1.0
Test Accuracy: 0.9517142857142857
Cross-Validation Accuracy (Mean/Std): 0.930 (0.022)
RBF SVM WITH SIGMA = 2^-3
Num Support Vectors: 282
Train Accuracy: 0.9975
Test Accuracy: 0.9728571428571429
Cross-Validation Accuracy (Mean/Std): 0.972 (0.018)
RBF SVM WITH SIGMA = 2^-4
Num Support Vectors: 155
Train Accuracy: 0.9925
Test Accuracy: 0.9711428571428572
Cross-Validation Accuracy (Mean/Std): 0.972 (0.018)
RBF SVM WITH SIGMA = 2^-5
Num Support Vectors: 105
Train Accuracy: 0.9925
Test Accuracy: 0.9705714285714285
Cross-Validation Accuracy (Mean/Std): 0.972 (0.018)
RBF SVM WITH SIGMA = 2^-6
Num Support Vectors: 96
Train Accuracy: 0.98
Test Accuracy: 0.9737142857142858
Cross-Validation Accuracy (Mean/Std): 0.968 (0.013)
RBF SVM WITH SIGMA = 2^-7
Num Support Vectors: 111
Train Accuracy: 0.9725
Test Accuracy: 0.9747142857142858
Cross-Validation Accuracy (Mean/Std): 0.963 (0.014)
RBF SVM WITH SIGMA = 2^-8
Num Support Vectors: 136
Train Accuracy: 0.97
Test Accuracy: 0.9744285714285714
Cross-Validation Accuracy (Mean/Std): 0.963 (0.014)
```

Figure 1.6: Twonorm Dataset Output.

Looking at the output for the twonorm dataset (Figure 1.6), we notice some similarities and differences compared to the banana dataset. This time, the linear SVM does a much better job of predicting with training accuracy of 0.99 and test accuracy of 0.97. The RBF SVM also does much better on average, but it maintains a similar trend where the accuracy of the RBF SVM depends greatly on the value of sigma. When sigma = 2^-1, the training accuracy is 1.0 while the test accuracy is 0.51, once again indicating overfitting. However, when sigma = 2^-5, the training accuracy is 0.99 and the test accuracy is 0.97, once again out-performing the linear SVM (before rounding the RBF SVM is higher).

Waveform Dataset

```
LINEAR SVM
Num Support Vectors: 85
Train Accuracy: 0.93
Test Accuracy: 0.8626086956521739
RBF SVM WITH SIGMA = 2^-1
Num Support Vectors: 400
Train Accuracy: 1.0
Test Accuracy: 0.6706521739130434
Cross-Validation Accuracy (Mean/Std): 0.670 (0.066)
RBF SVM WITH SIGMA = 2^-2
Num Support Vectors: 390
Train Accuracy: 1.0
Test Accuracy: 0.8
Cross-Validation Accuracy (Mean/Std): 0.802 (0.041)
RBF SVM WITH SIGMA = 2^-3
Num Support Vectors: 262
Train Accuracy: 0.985
Test Accuracy: 0.8904347826086957
Cross-Validation Accuracy (Mean/Std): 0.925 (0.025)
RBF SVM WITH SIGMA = 2^-4
Num Support Vectors: 166
Train Accuracy: 0.97
Test Accuracy: 0.8965217391304348
Cross-Validation Accuracy (Mean/Std): 0.935 (0.012)
RBF SVM WITH SIGMA = 2^-5
Num Support Vectors: 146
Train Accuracy: 0.955
Test Accuracy: 0.8939130434782608
Cross-Validation Accuracy (Mean/Std): 0.938 (0.008)
RBF SVM WITH SIGMA = 2^-6
Num Support Vectors: 150
Train Accuracy: 0.945
Test Accuracy: 0.8860869565217391
Cross-Validation Accuracy (Mean/Std): 0.925 (0.008)
RBF SVM WITH SIGMA = 2^-7
Num Support Vectors: 165
Train Accuracy: 0.93
Test Accuracy: 0.8817391304347826
Cross-Validation Accuracy (Mean/Std): 0.912 (0.008)
RBF SVM WITH SIGMA = 2^-8
Num Support Vectors: 190
Train Accuracy: 0.9225
Test Accuracy: 0.8730434782608696
Cross-Validation Accuracy (Mean/Std): 0.910 (0.009)
```

Figure 1.7: Waveform Dataset Output.

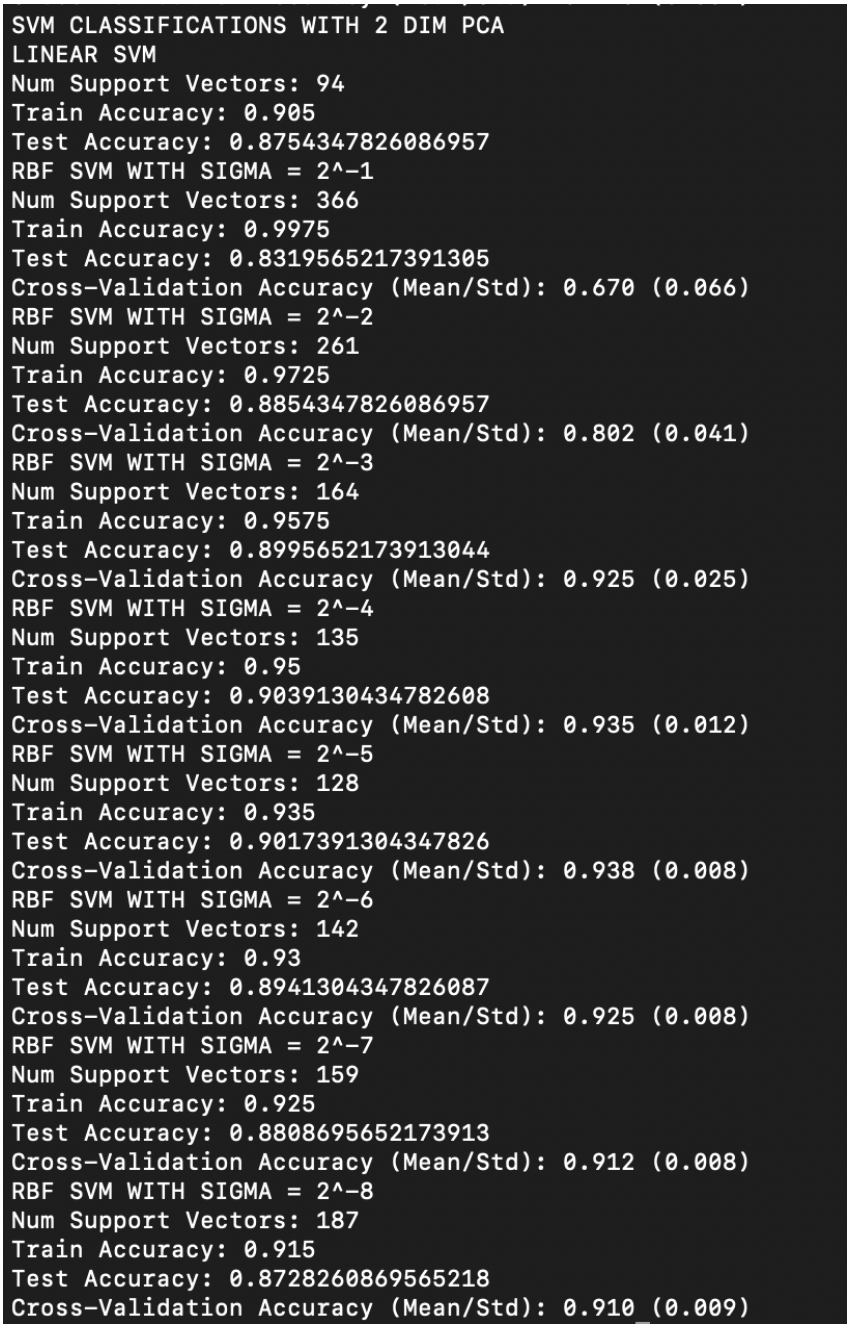


Figure 1.8: Waveform Dataset with 2-Dimensional PCA Output.

Looking at the output for the waveform dataset (Figure 1.7), we once again notice the same trends. The linear SVM has training/test accuracy of 0.93/0.86, while the RBF SVM accuracy ranges from 0.99/0.89 to 1.0/0.67 depending on the value of sigma. With an optimal value of sigma, 2^-3, the RBF SVM outperforms the linear, but with a poor value of sigma, it does worse.

Furthermore, looking at the PCA-reduced output (Figure 1.8), we can see that despite reducing the data to two dimensions, we preserved most of the information. All accuracies stayed above 0.85, and the variance between original and PCA-reduced for each data point is very small.

Task Two: Neural Networks

Intro to Neural Networks

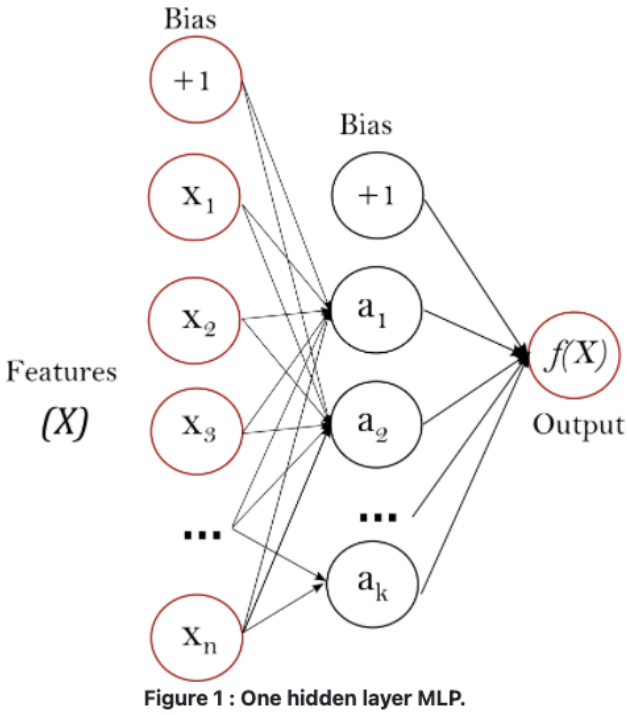


Figure 2.1: Three Layer Neural Network.

$$S(x) = \frac{1}{1 + e^{-x}}$$

Figure 2.2: Sigmoid Function.

Neural Network (NN) is a classification technique that works by incrementally building a complex function that ultimately determines the class of each data point. The network starts with the input layer which is just the raw dataset. From there, we see a variable number of hidden layers. At each layer, an activation function is applied to the data, changing the data to better fit the criteria of the prediction, before being passed on to the next layer. Finally, after passing through the output layer, the neural network generates the classification for the data points (Figure 2.1). Note that each node in a neural network is called a neuron.

In this task, we implement a 3-layer neural network (which has one hidden layer) with a sigmoidal activation function (Figure 2.2). For this task, we are trying to determine how many neurons the hidden layer should have to produce the most accurate classification.

Step-By-Step Implementation

For this task, I created one python file for each dataset we need to process: banana_nn.py, twonorm_nn.py, and waveform_nn.py. For each dataset, we first need to import all the necessary libraries. For this task, I leverage sklearn's NN classifier so I don't have to implement one from scratch. Then, we need to initialize all the training and test datasets. Then, I loop through all the possible values for the number of neurons in the hidden layer, keeping track of which value produces the most accurate result for training and test data. Finally, I print the results of the classifier with optimal neuron values to analyze its accuracy.

Banana Dataset

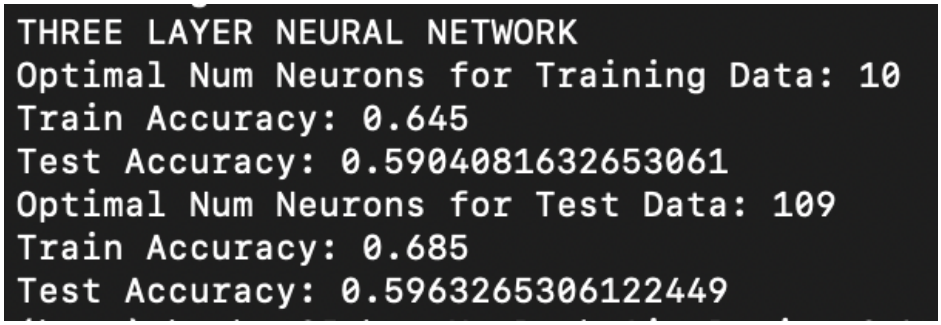
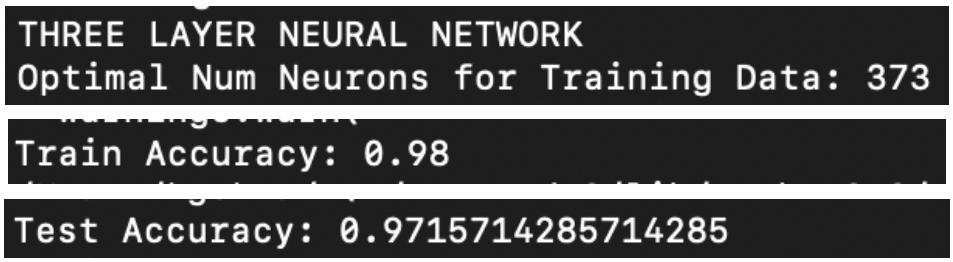


Figure 2.3: Banana Dataset Output.

Looking at the output for the banana dataset (Figure 2.3), we can see that the 3-layer neural network produced accuracies ranging between 0.60 and 0.70 even with an optimal number of neurons. These values are similar to the performance of the linear SVM with training/test accuracies of 0.70/0.61. However, the optimal neural network vastly underperforms compared to the optimal RBF SVM with training/test accuracies of 0.96/0.86. One reason for this might be because the neural network model has a lot of fine-tuning parameters. For this task, we were only told to analyze one of them, the number of neurons, so I left all the other ones at their default values set by sklearn when you initialize the model. As a result, it's possible these default values don't work well with the banana dataset, causing the accuracy to be relatively low, even in the "optimal case" (optimal only in respect to number of neurons).

Twonorm Dataset



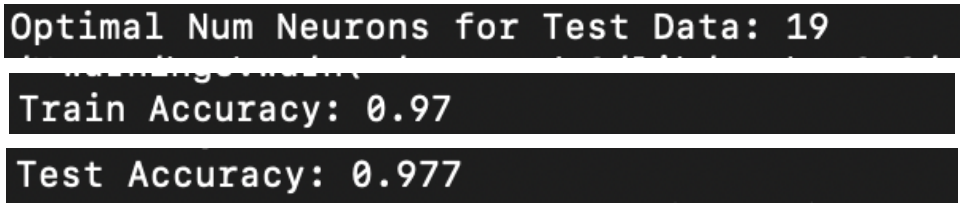


Figure 2.4: Twonorm Dataset Output.

Looking at the output for the twonorm dataset (Figure 2.4), we see much better expected values, with training/test accuracy ranging between 0.97 and 0.98. Similar to the banana dataset, these values are very similar to the linear SVM training/test accuracies of 0.99/0.96. The RBF SVM only slightly outperforms in the optimal case. Analyzing banana and twonorm, we can hypothesize that an optimal neural network performs about as good as the linear SVM but is outperformed by an optimized RBF SVM. Next we will analyze the waveform dataset to see if it fits our hypothesis.

Waveform Dataset

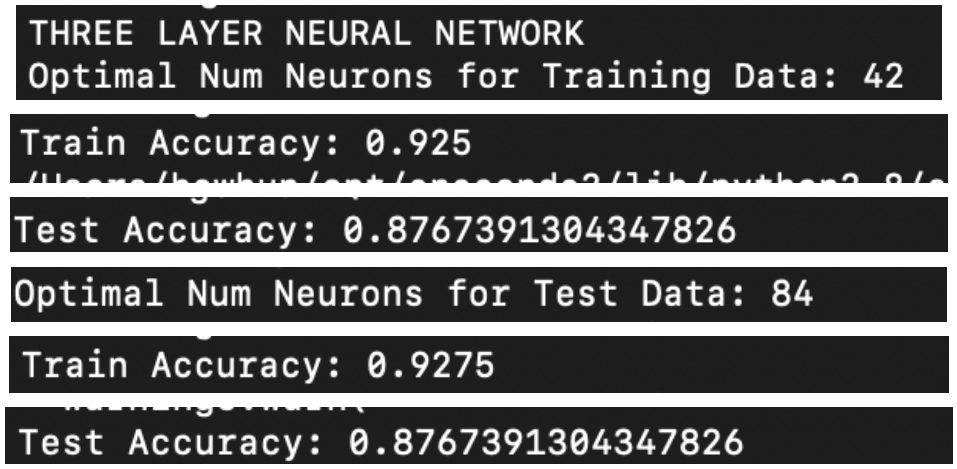


Figure 2.5: Waveform Dataset Output.

Looking at the waveform dataset output (Figure 2.5), we can see training/test accuracies around 0.93/0.87 in the optimal case. This is similar to the linear SVM training/test accuracies of 0.93/0.86. Once again, the optimized RBF SVM outperforms with training/test accuracies of 0.99/0.89. In conclusion, we can make the connection that an optimized neural network produces similar results as a linear SVM but is outperformed by an optimized RBF SVM.

Task Three: Clustering

Intro to Clustering

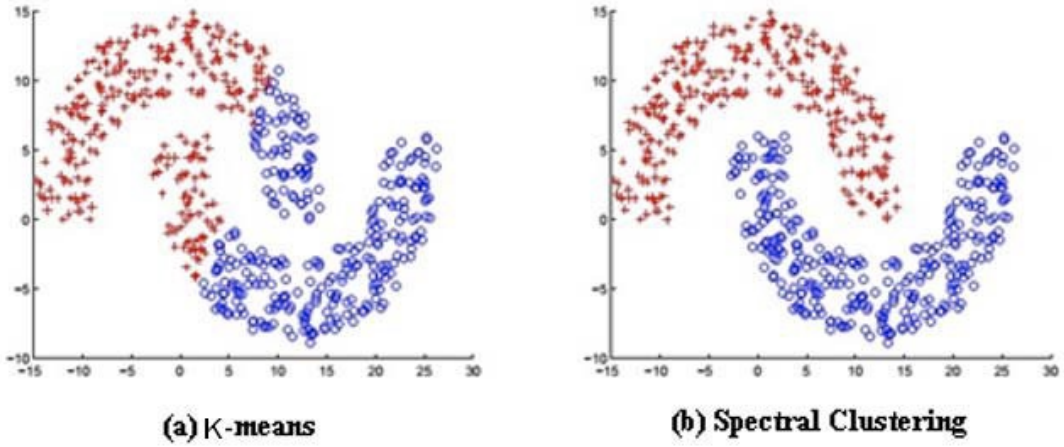


Figure 3.1: KMeans vs. Spectral Clustering.

$$\text{if } i \neq j, \quad w_{ij} = exp\big(-||x_i - x_j||^2/10\big), \forall i, j \text{ else } w_{ii} = 0, \forall i$$

Figure 3.2: Spectral Clustering Similarity Measure.

KMeans clustering is a classification technique by which each data point is grouped into a cluster depending on which cluster's centroid that data point is closest to. As such, KMeans clustering is often considered "absolute" clustering. On the other hand, Spectral Clustering clusters data points together based on their affinity or similarity instead of absolute location. As such, Spectral Clustering is considered "relative" clustering (Figure 3.1). The similarity of two points in Spectral Clustering is determined by a similarity measure that's user-generated and passed into the model. For this task, we will be using the similarity measure defined in the task description (Figure 3.2).

The Gaussian Mixture Model is another clustering technique that uses estimation of the density of the dataset to split the dataset into clusters. By leveraging the weights, means, and covariances of each cluster, this model uses the Expectation-Maximization algorithm to update each cluster.

In this task, we change the number of clusters and the clustering technique used on each dataset to analyze the effects both of these variables have on classification accuracy.

Step-By-Step Implementation

For this task, I created two python files for each dataset we need to process (one for Kmeans and Spectral Clustering, and one for Gaussian Mixture): cluster_within_cluster_(clusters/gaussian_mix).py, two_spirals_(clusters/gaussian_mix).py, crescent_and_the_full_moon_(clusters/gaussian_mix).py, and waveform_clusters.py. For each dataset, we first need to import all the necessary libraries. For this task, I leverage sklearn's KMeans, Spectral Clustering, and Gaussian Mixture classifiers so I don't have to implement them from scratch. Then, we need to initialize all the training and test datasets.

For the first part of this task, I calculate the affinity matrix for the dataset so I can pass it into sklearn's Spectral Clustering model. Then, I initialize four models: KMeans with 2 and 4 clusters, and Spectral Clustering with 2 and 4 clusters. After initializing all four models, I plot the results of these clustering algorithms to analyze their output.

For the second part of this task, I do the same initializations of the four models, but this time instead of plotting them, I use them to predict training and test data classes, printing the resulting accuracies.

For the third task, I initialize a Gaussian Mixture model, and this time I both plot the resulting classifications in a scatter plot and also print the accuracy on training and test data for analysis.

Two-Spirals Clustering

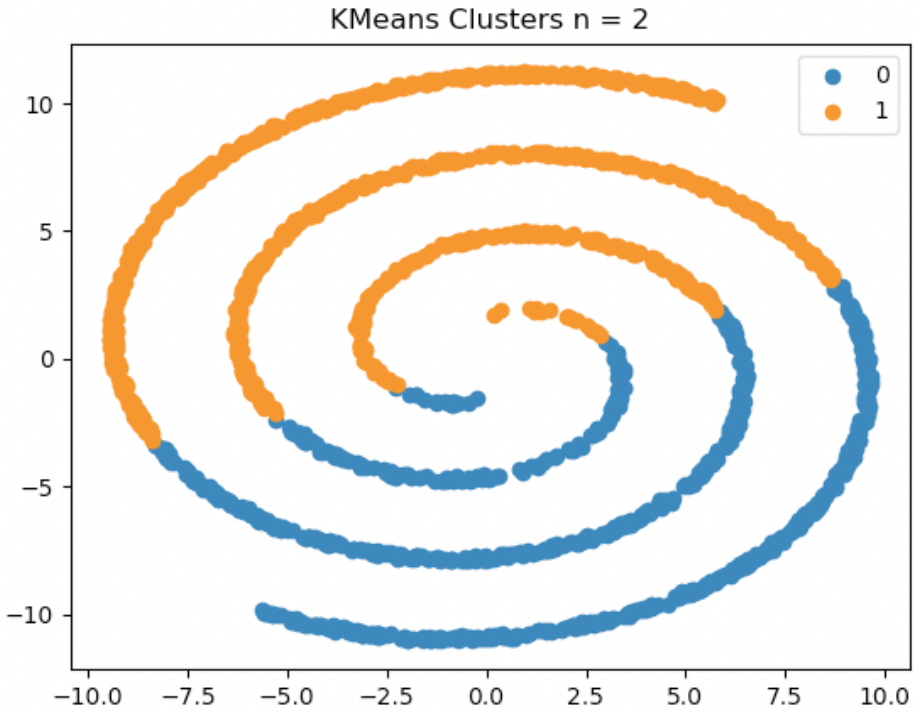


Figure 3.3: KMeans Clusters n = 2.

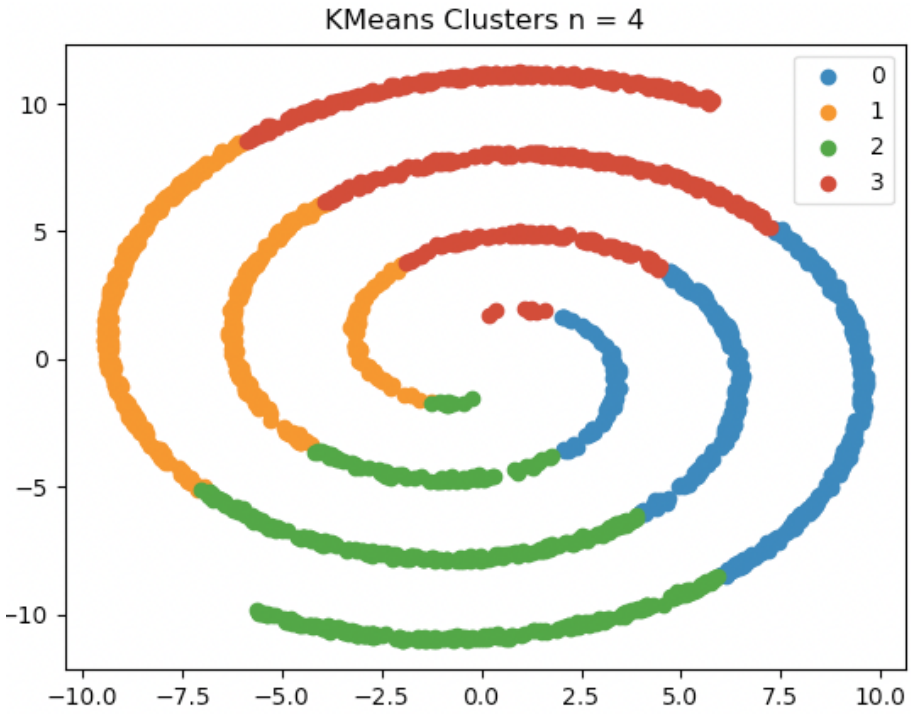


Figure 3.4: KMeans Clusters n = 4.

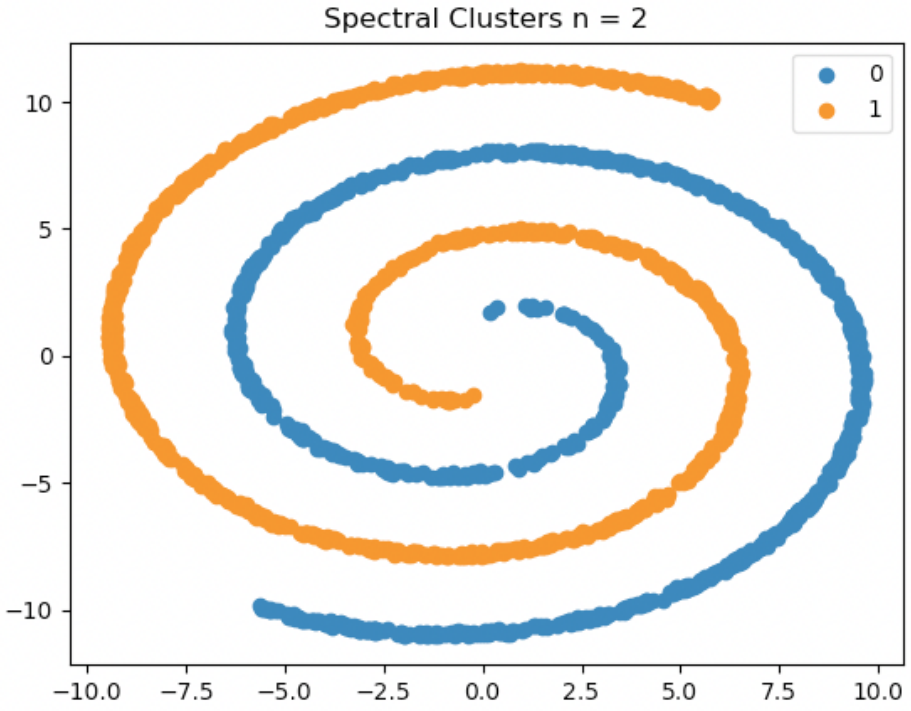


Figure 3.5: Spectral Clusters n = 2.

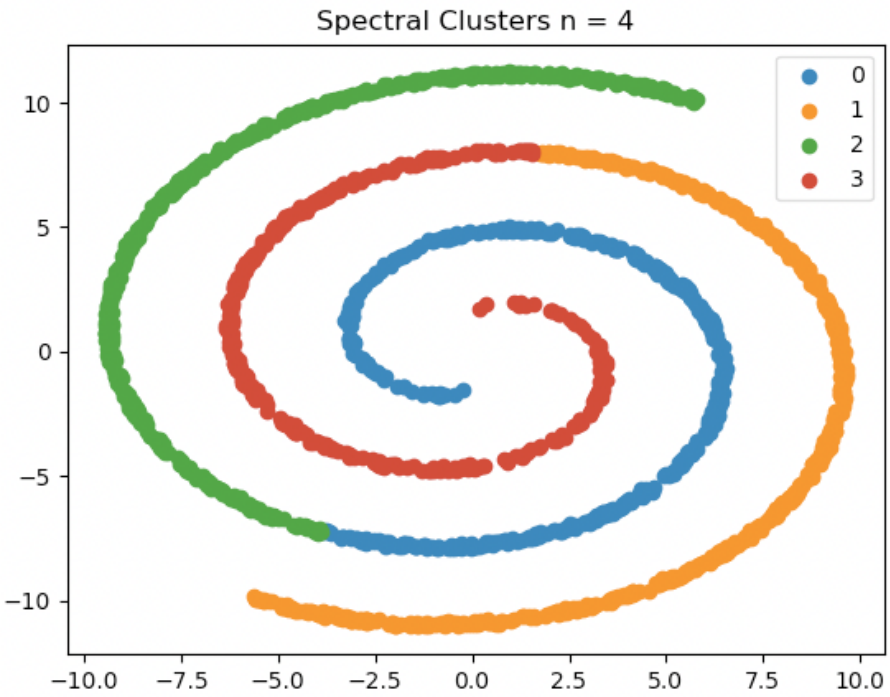


Figure 3.6: Spectral Clusters n = 4.

Looking at the plots for KMeans and Spectral Clusters for two-spirals dataset (Figures 3.3-3.6), we can see that KMeans splits the graph into two halves and four quadrants when n = 2 and 4 respectively. However, it doesn't account for the actual shape of the data or the similarity that points might have with one another.

On the other hand, we can see that Spectral Clusters actually group data points of each spiral together. When n = 2, Spectral Clusters correctly identifies each spiral. When n = 4, Spectral Clusters break each spiral into the front and back ends of each spiral, creating four clusters. Looking at all four graphs side by side, we can conclude that Spectral Clusters with n = 2 does the best job of classifying the data and creating the two spirals we were expecting to see.

Cluster-Within-Cluster Clustering

Figure 3.7: KMeans Clusters n = 2.

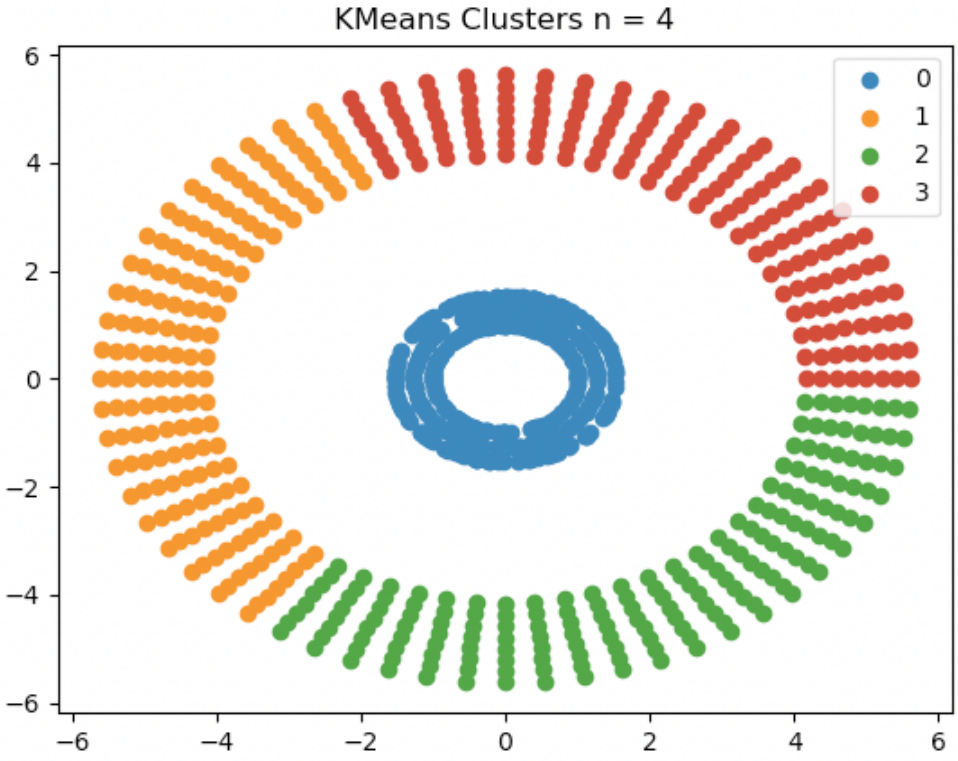


Figure 3.8: KMeans Clusters n = 4.

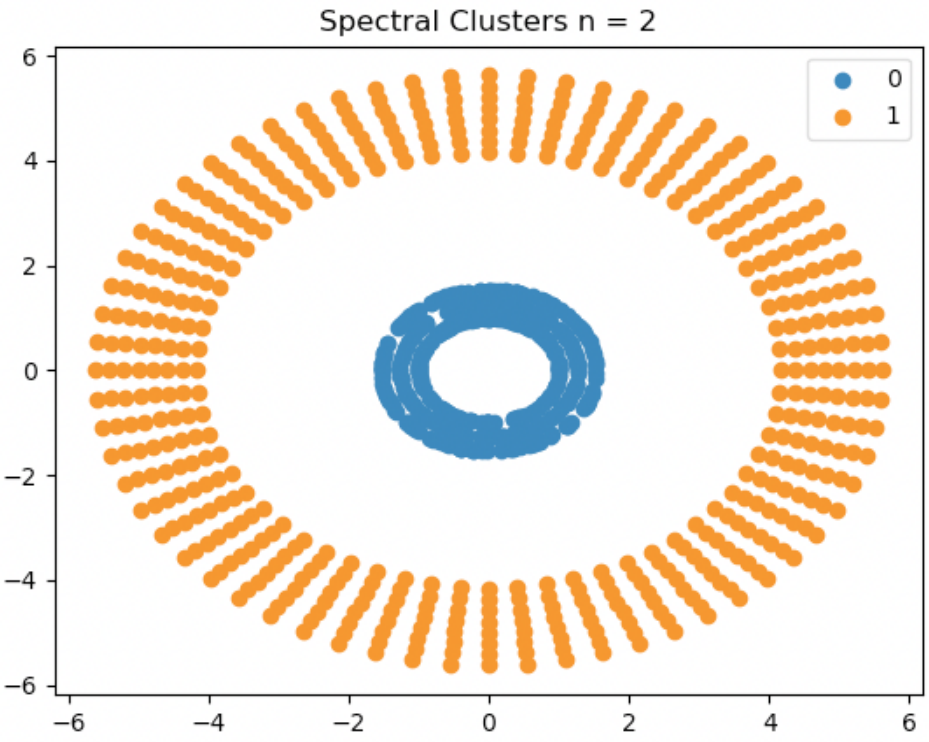


Figure 3.9: Spectral Clusters n = 2.

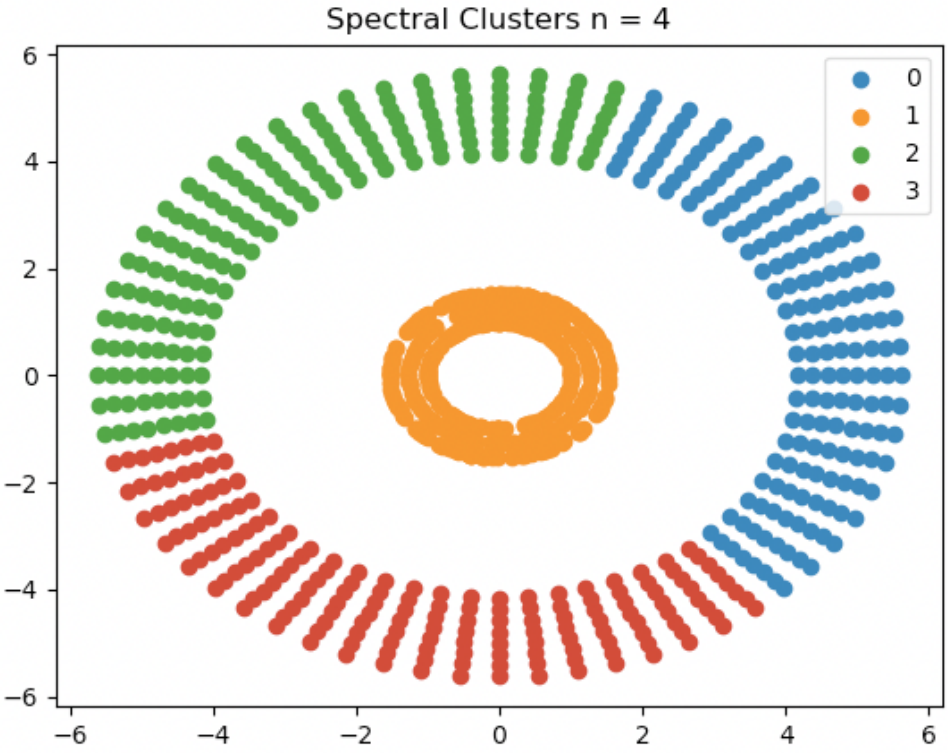


Figure 3.10: Spectral Clusters n = 4.

Looking at the plots for KMeans and Spectral Clusters for two-spirals dataset (Figures 3.7-3.10), we can see KMeans once again splits the graph in half when n = 2. However, when n = 4, instead of splitting the graph into four quadrants, KMeans actually has one cluster as the middle circle then splitting the outer circle in thirds.

On the other hand, Spectral Clustering correctly identifies the inner and outer circles when n = 2. When n = 4, Spectral Clustering also identifies the inner circle and splits the outer circle into thirds, similar to the KMeans n = 4 case. Once again, Spectral Clustering n = 2 correctly classifies the data we were expecting to see.

Crescent-and-the-Full-Moon Clustering

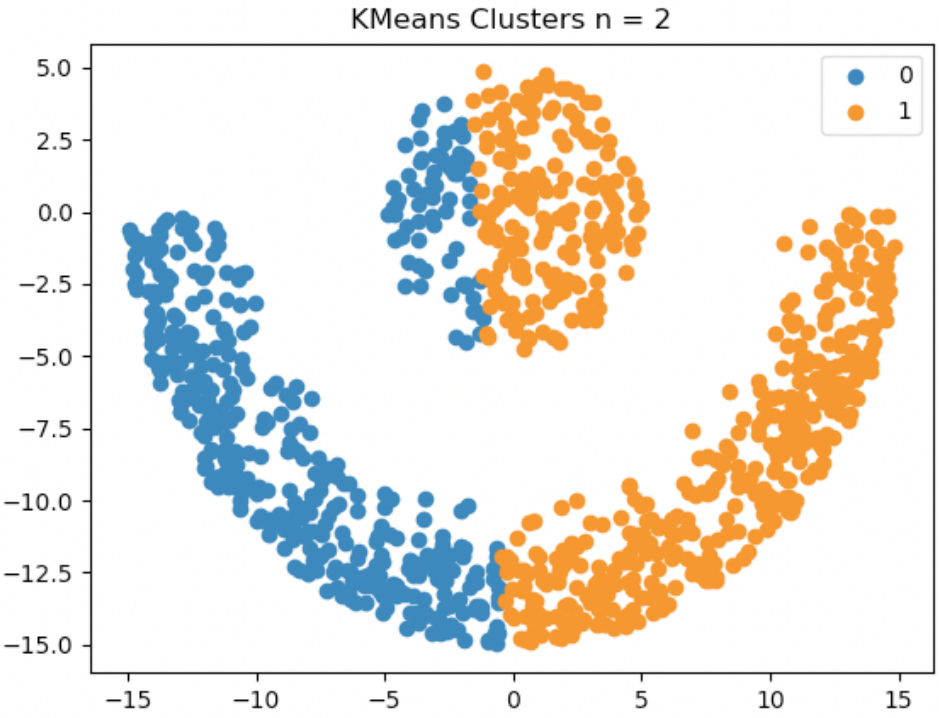


Figure 3.11: KMeans Clusters n = 2.

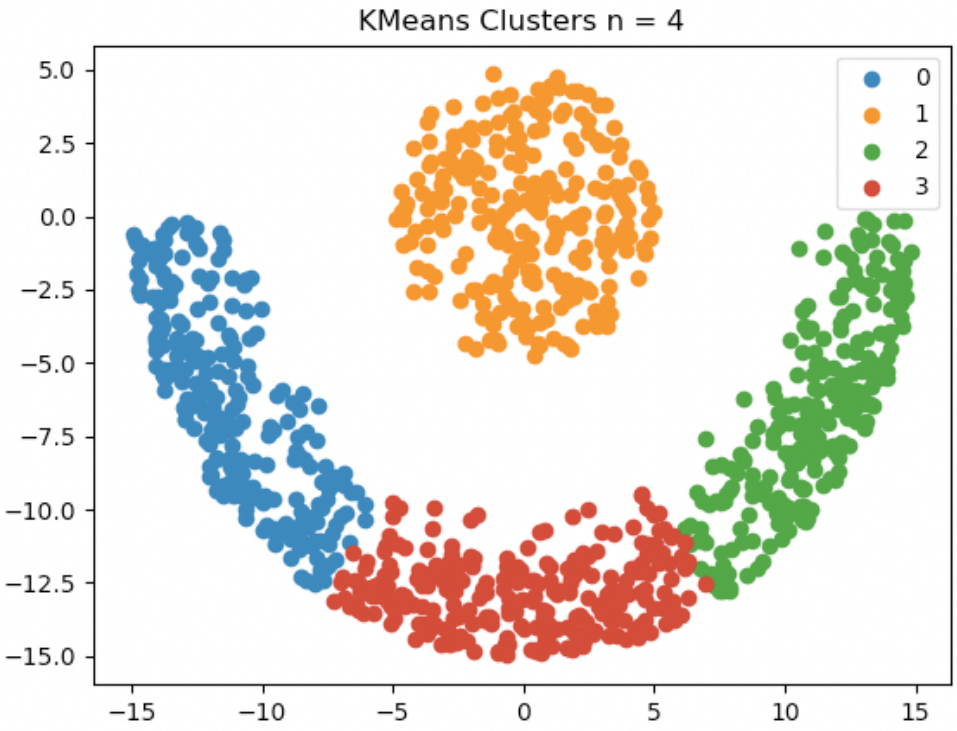


Figure 3.12: KMeans Clusters n = 4.

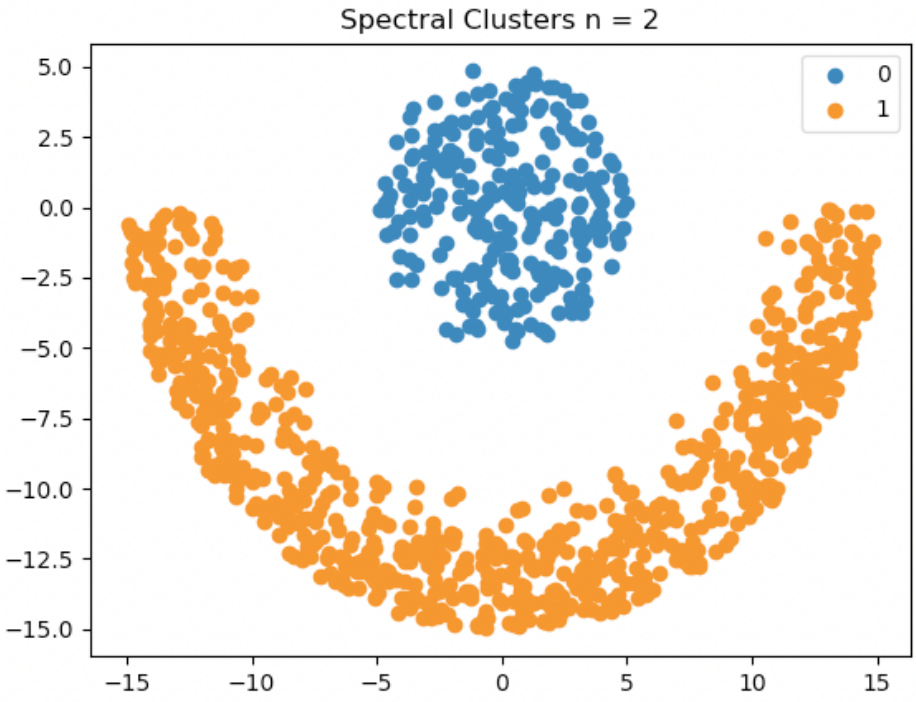


Figure 3.13: Spectral Clusters n = 2.

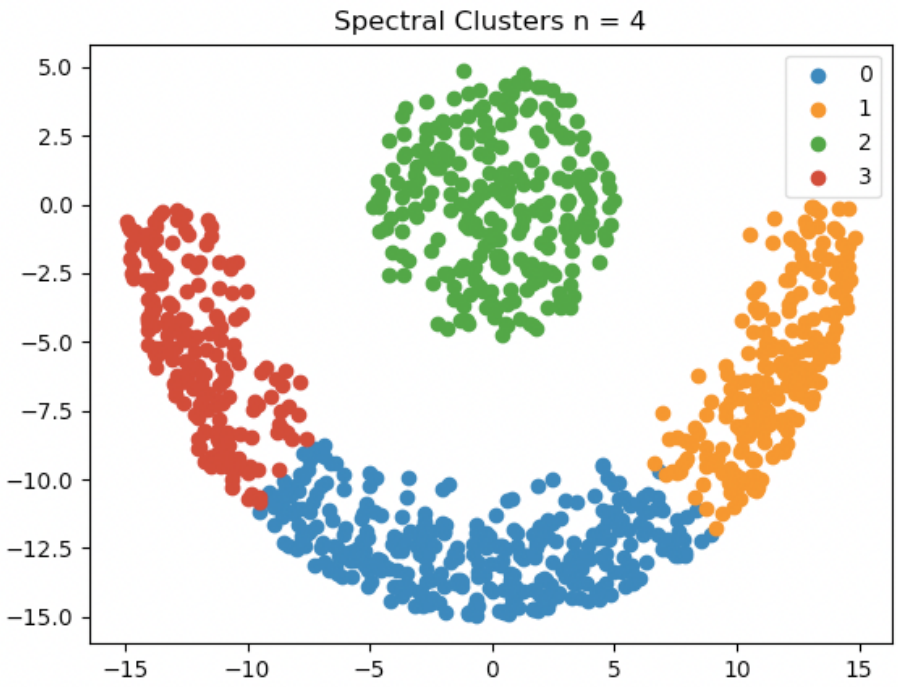


Figure 3.14: Spectral Clusters n = 4.

Looking at the plots for KMeans and Spectral Clusters for two-spirals dataset (Figures 3.11-3.14), KMeans once again splits the graph in half when n = 2. Similar to the dataset above, when KMeans n = 4, it identifies the circle on top as a cluster, then it splits the crescent into three clusters.

Also similar to the above dataset, Spectral Clustering correctly identifies the circle and the crescent when n = 2. When n = 4, it has the same behavior as KMeans, identifying the circle then splitting the crescent into thirds. We once again see that Spectral Clustering n = 4 correctly classifies the data of this dataset.

Waveform Clusters

```
WAVEFORM CLUSTERS
KMeans Clusters n = 2
Train Accuracy:  0.8075
Test Accuracy:  0.8017391304347826
Spectral Clusters n = 2
Train Accuracy:  0.695
Test Accuracy:  0.7865217391304348
```

Figure 3.15: Waveform Dataset Output

Looking at the outputs for the waveform dataset (Figure 3.15), we can see that KMeans n = 2 actually outperforms Spectral Clustering n = 2. Comparing that to the above trends, we would expect Spectral Clustering n = 2 to be better but that turned out to be false.

Despite KMeans being better, it still has a lower accuracy than the other two models we analyzed above. Both SVM and Neural Networks produced accuracies above 0.85, with optimal cases going above 0.95. This showcases the effectiveness of clustering vs other classification techniques, at least for this particular dataset.

Two-Spirals Gaussian Mixture

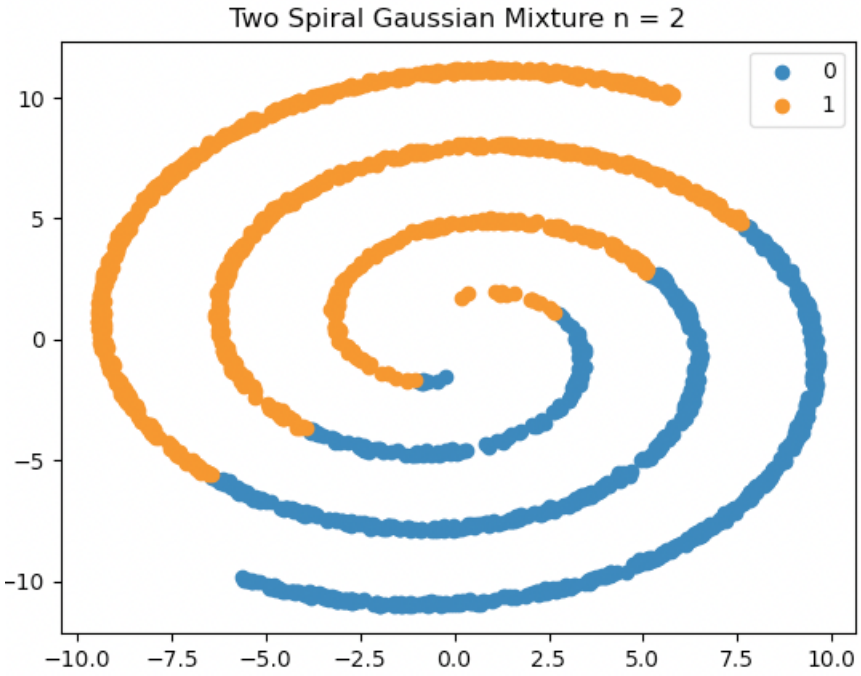


Figure 3.16: Two Spiral Gaussian Mixture n = 2.

```
GAUSSIAN MIXTURE
Weights:
[0.4760927 0.5239073]
Means:
[[ 2.95577261 -5.15916657]
 [-2.48257683  4.50977617]]
Covariances:
[[[23.00539413 11.25662896]
  [11.25662896 18.47830997]]

 [[24.30129662  9.93084256]
  [ 9.93084256 20.58511771]]]
```

Figure 3.17: Two Spiral Output.

Looking at the two spiral Gaussian Mixture plot (Figure 3.16), we can see that it visually resembles the KMeans n = 2 plot by simply dividing the graph in half. Furthermore, the output of the distribution can be seen in Figure 3.17.

Cluster-Within-Cluster Gaussian Mixture

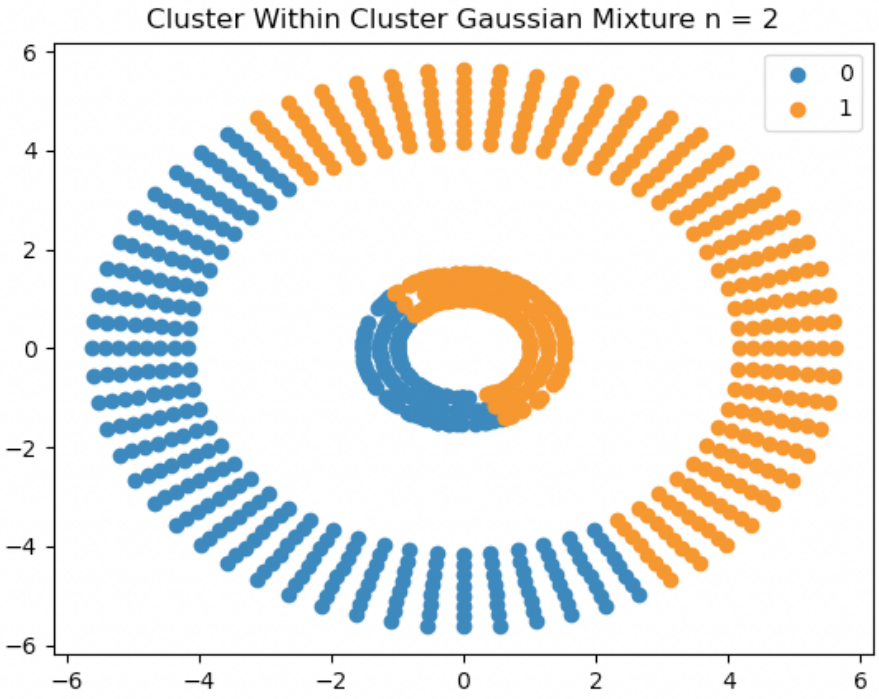


Figure 3.18: Cluster Within Cluster Gaussian Mixture n = 2.

```
Weights:
[0.46477818 0.53522182]
Means:
[[-1.37917565 -0.92419879]
 [ 1.18136461  0.87912216]]
Covariances:
[[[ 5.2663538  -1.14466048]
  [-1.14466048  6.29584194]]

 [[ 5.28437394 -1.12259034]
  [-1.12259034  5.93963613]]]
```

Figure 3.19: Cluster Within Cluster Output.

Looking at the cluster-within-cluster Gaussian Mixture plot (Figure 3.18), we can see that it visually also resembles the KMeans n = 2 plot by simply dividing the graph in half. Furthermore, the output of the distribution can be seen in Figure 3.19.

Crescent-and-the-Full Moon Gaussian Mixture

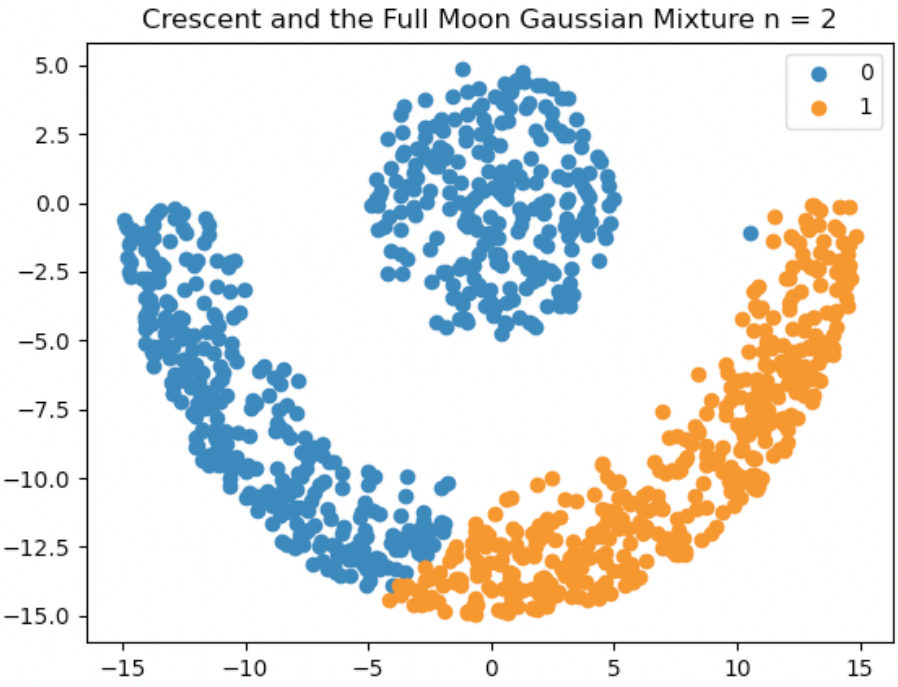


Figure 3.20: Crescent and the Full Moon Gaussian Mixture.

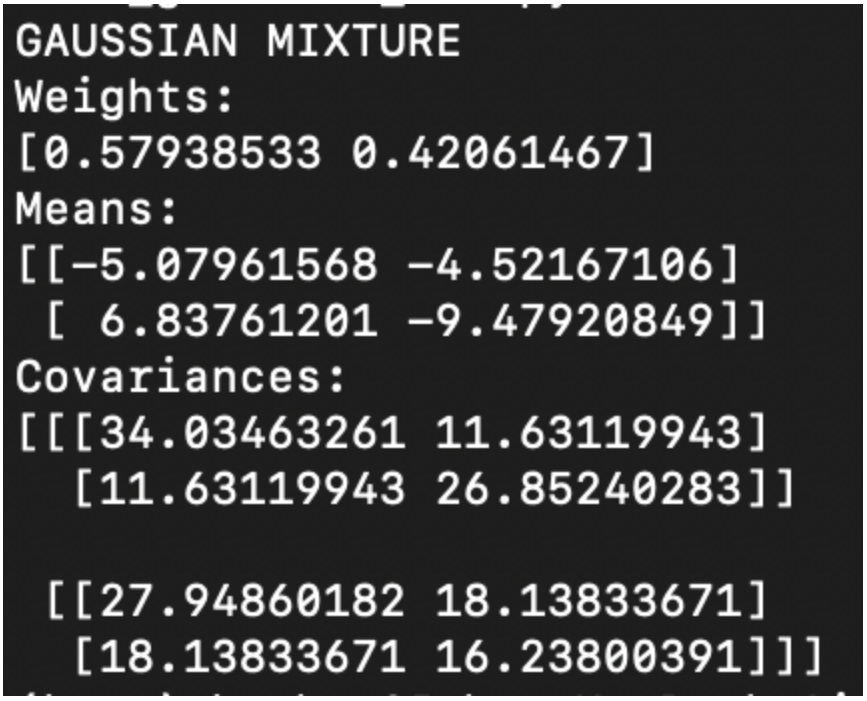


Figure 3.21: Crescent and the Full Moon Output.

Looking at the crescent and the full moon Gaussian Mixture plot (Figure 3.20), we can see that it visually also resembles the KMeans n = 2 plot by simply dividing the graph in half. Furthermore, the output of the distribution can be seen in Figure 3.21.