PRML ASSIGNMENT-5

QUESTION-1

TASK-1 : computeCentroid Function

The computeCentroid function uses RGB values to describe a set of 3-dimensional characteristics and determines their mean. The implementation ensures that each RGB component is averaged independently by appropriately using NumPy's mean function along the designated axis. This method efficiently computes the centroid of the provided features, which complies with the job criteria.

TASK-2 : mykmeans Function

The K-means clustering method must be implemented from scratch via the mykmeans function. The number of clusters (k), the data matrix X containing pixel values, and the starting centroids are the inputs for this function. After the K-means algorithm is completed, it returns the final cluster centres.

The function starts by taking the structure of the data matrix and extracting the number of data points (m). After that, iterating through a certain number of iterations (max_iter) and updating the centroids each time by calculating the mean of the data points allocated to each centroid, it does this process.

The function uses Euclidean distance to determine the separation between each data point and the centroids for each iteration. Every data point is assigned to the closest centroid, and the data points associated to each centroid are gathered and stored in a dictionary (centroid_rgbs).

The function updates the centroids by calculating the mean of the data points assigned to each centroid once all data points have been assigned to centroids. The centroids are allowed to gradually converge to their final positions by repeating this process for the designated amount of iterations.

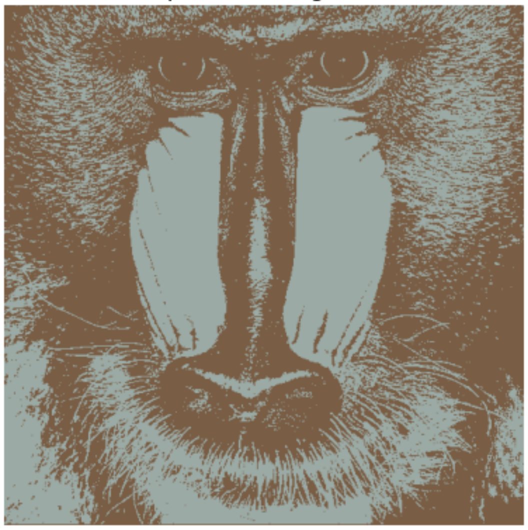
After applying the K-means algorithm, the function produces the final centroids, which stand for the cluster centres.

TASK-3 : Image compression with different k values

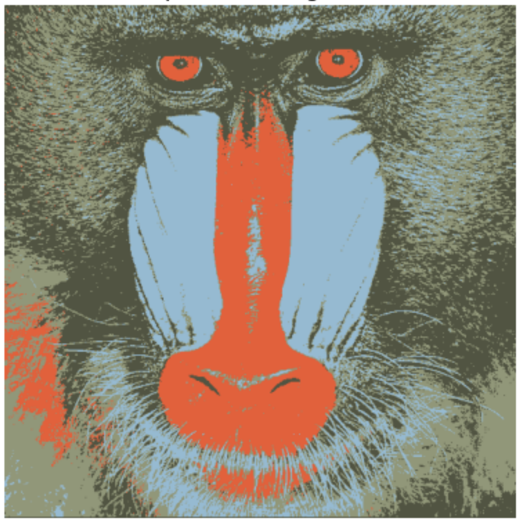
We used the K-means clustering algorithm to compress the images in this task, varying the value of k to indicate the number of clusters. In the beginning, we defined a list called num_clusters_list that held the compression-related values for k. We used the init_centroids function to initialise centroids for each value of k in the list, and our own mykmeans implementation was used to carry out K-means clustering. As a result, we were able to determine the final centroids that represented the image's colour groupings.

Next, we used the update_image function to update the image data with the final centroids. In this stage, the nearest centroid for each pixel in the image was determined using the Euclidean distance. We efficiently reduced the size of the image without sacrificing any of the vital colour information by substituting the colour of each pixel for that of its matching centroid. Lastly, we used Matplotlib to show the compressed images for various values of k. This gave us a visual representation of how changing the number of clusters affects the quality of the compression and the appearance of the final image.

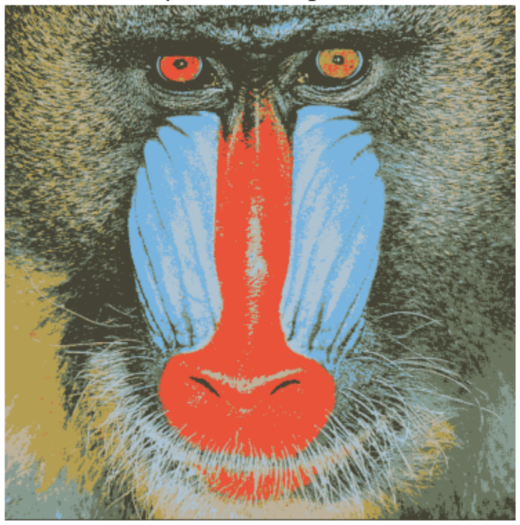
Compressed Image (k=2)



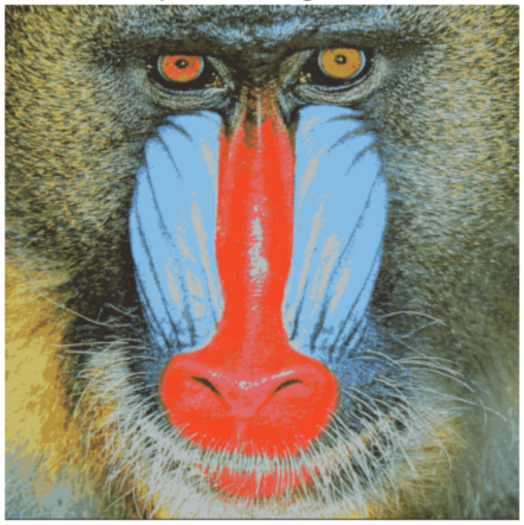
Compressed Image (k=4)



Compressed Image (k=8)



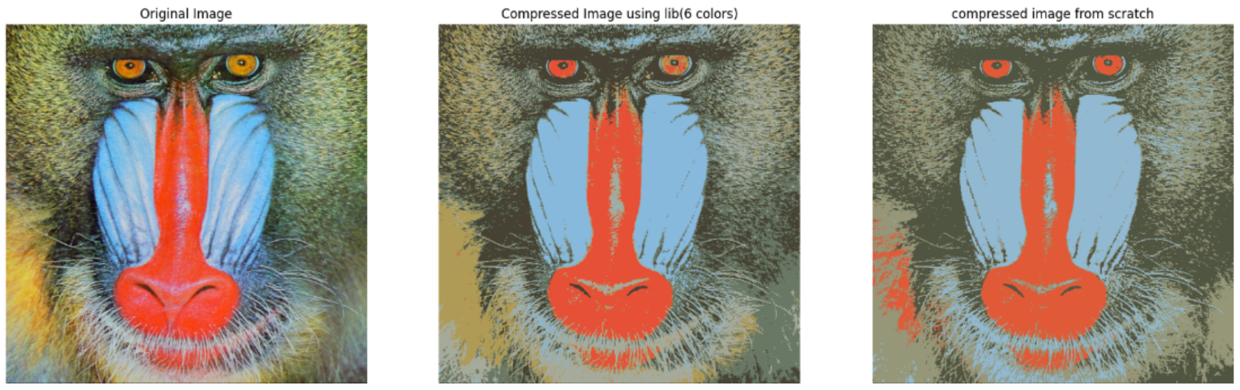
Compressed Image (k=20)



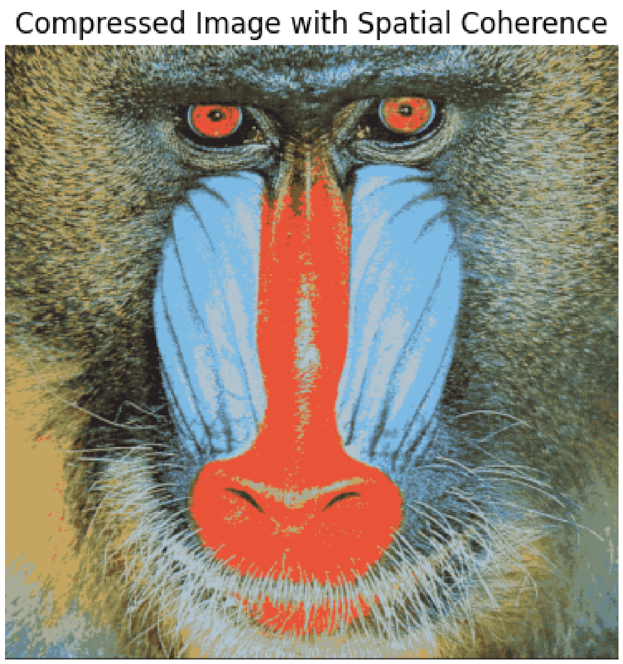
TASK-4 : Comparison with sklearn’s K-mean implementation

We used the KMeans class in scikit-learn to do K-means grouping with a certain number of groups, in this case six. After the reshaped picture data was fitted to the KMeans instance, the labels and centres were found. We put the compressed image back together by giving each pixel the colour of its associated centroid using the labels and centroids.

We put the original image, the compressed image made with scikit-learn's K-means implementation, and the compressed image made by our custom implementation next to each other so that you could see the differences. This let us see if the two methods had any changes in image quality, colour representation, or compression artefacts. By looking at the visual result, we were able to tell how well and how differently each implementation compressed the picture while keeping important visual features.


Original Image
Compressed Image using lib(6 colors)
compressed image from scratch

TASK-5 : Spatial coherence



When you use spatial coherence in image compression, you have to give pixels to clusters that are close to each other both in terms of colour similarity and spatial proximity. 'mykmeans_with_spatial_coherence' is a modified version of the K-means clustering method that we created to make this idea work. This tool finds the colour and spatial distances between pixels and their centres. We make sure that pixels close together in the original picture are more likely to be assigned to the same cluster by combining these distances with a certain spatial weight parameter. Keeping local structures helps cut down on artefacts like noise or colour bleeding, which improves the quality of the compressed picture as a whole.

After putting the plan into action and looking at the final image, we saw a big change in how well local structures were preserved and how many compression artefacts were gone. It is now more likely for pixels that were close together in the original picture to be in the same cluster. This makes the transitions smoother and better shows how spaces are connected. For example, adding spatial coherence to a compressed picture makes it clearer and more accurate, showing that this method works to keep the original image's integrity during compression.

QUESTION-2

Colab link : Untitled2.ipynb

TASK-1(A) :

We got the Iris dataset from the sklearn.datasets module and used the load_iris() method with as_frame=True to get the data back in a pandas DataFrame format for Task 1(a). Measurings of sepal length, sepal width, petal length, and petal width for three types of iris are in the collection. These are Iris setosa, Iris versicolor, and Iris virginica.

To get the data ready for binary classification, the target column was filtered so that it only had 0 and 1 values. This chose only the "setosa" and "versicolor" classes. "Petal length (cm)" and "petal width (cm)" were chosen as the predictor factors X, and "class label" (y) was the label to be used.

The feature data X was normalised using the StandardScaler() from sklearn.preprocessing before it was split into training and test sets. This makes the features scaled so that they have a mean of zero and a variance of one, which is helpful for many machine learning methods.

Finally, train_test_split() from sklearn.model_selection was used to divide the normalised X and y into training and test sets. This test set has a size of 0.2 (20%) and a random_state of 42 to make it easy to repeat. The training data will be used to teach the SVM models new things, and the test data will be used to check how well the models did in the future.

TASK-1(B) :

To demonstrate where the learnt Linear Support Vector Classifier (LinearSVC) model made its decisions, a meshgrid of feature values was created for the scaled petal length and width data. This meshgrid was used to predict with the trained LinearSVC model's predict technique. This categorised each feature space point.

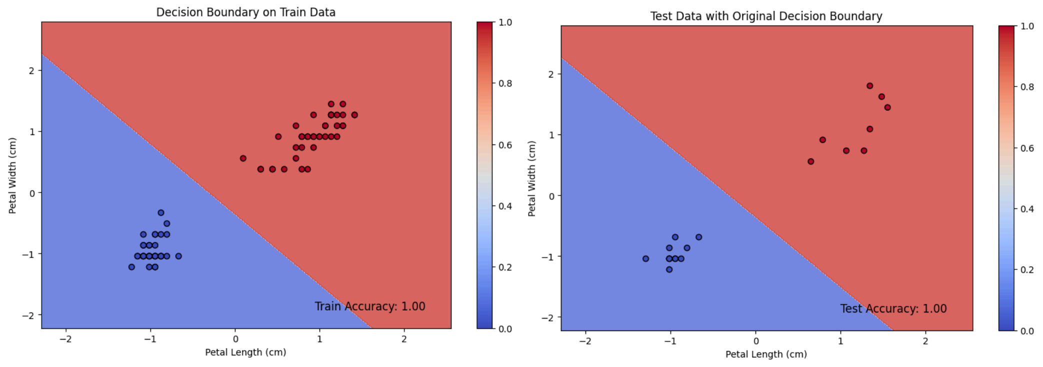
After drawing the decision border with plt.contourf to colour the meshgrid points based on the model, a coolwarm colormap was used to distinguish the two groups. The training data points were stacked in a scatter plot, and their colours indicated their class.

Sklearn's accuracy_score determined the train's accuracy, which was displayed on the map. This generated a number for how successfully the model distinguishes setosa and versicolor in the training data by petal size.

A separate plot showed the test data scatter and the training data decision boundary. We tested how well the decision boundary applied to new test cases. Also revealed was the test's accuracy score.

These charts show how the LinearSVC model's linear decision boundary separates petal measurements into two classes in the projected 2D feature space. We measure performance using learn and test data accuracy ratings.

These visualisations and metrics are crucial to model analysis. They can assist with SVM model kernel and hyperparameter selection in later project tasks.



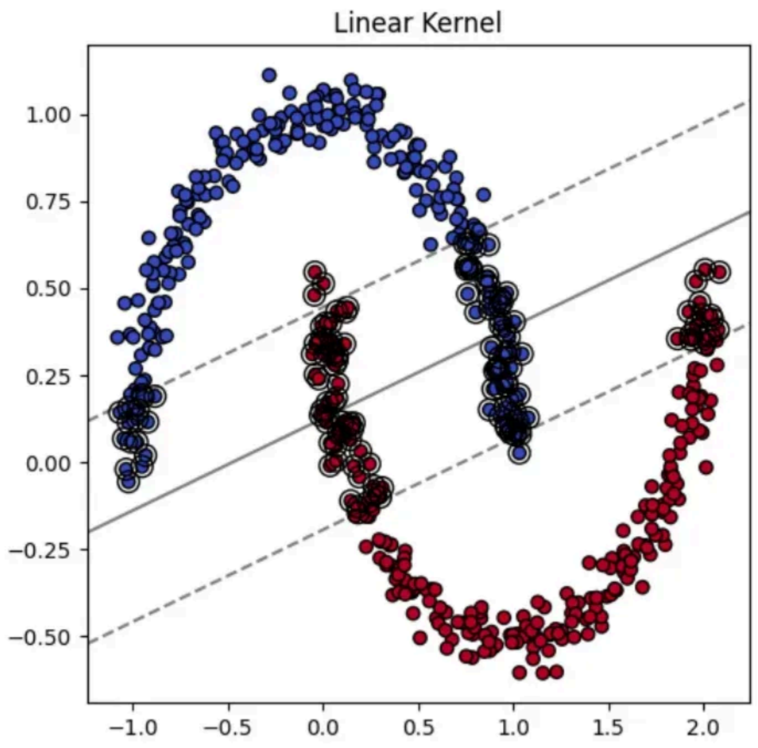
We expect the decision boundary to be a straight line since the LinearSVC model identifies the best linear hyperplane between the classes. Overlaid on the plot are blue circles for setosa samples and red circles for versicolor samples.

The plot shows that the LinearSVC model separated the two classes on the training data perfectly. The setosa class data points are all blue, while the versicolor samples are all red. The displayed train correctness of 1.00 or 100% confirms this.

TASK-2(A) :

We created a synthetic binary classification dataset with a moon-shaped decision border using the scikit-learn make_moons() method. The dataset has 500 data points (n_samples=500), and in order to induce some misclassifications, 5% noise (noise=0.05) was included. The reproducibility of the same dataset over several runs is guaranteed by the random_state=42.

TASK-2(B) :

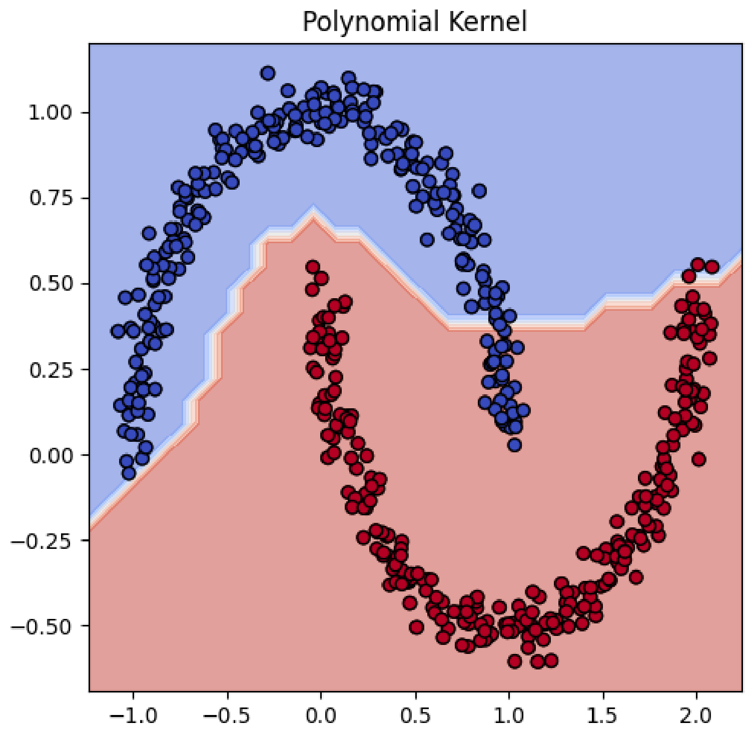


The decision boundary that was acquired by training a linear kernel SVM on the fictitious dataset produced by the make_moons() function is shown on the graph. The graphic makes it clear that the linear kernel SVM finds it difficult to distinguish between the two non-linearly separable moon-shaped classes.

The dashed line depicts the linear decision boundary, which passes through the centre of both moon-shaped clusters and results in a sizable number of misclassifications. For this dataset, the linear kernel can only identify a linear hyperplane that offers the best separation between the two classes; it is unable to identify the complex non-linear decision boundary.

The blue and red circles show the two classes' data points, which are mixed together and heavily overlap in some areas. This suggests that the linear kernel is insufficient to correctly categorise the samples. This restriction results from the linear kernel's inability to handle datasets with non-linear structures, which necessitate complex, non-linear decision boundaries. It can only handle data that can be divided into linear segments.

Employing non-linear kernels capable of mapping the data into higher-dimensional spaces, where the classes may become linearly separable, is important to improve the classification performance for datasets with non-linear decision boundaries, like the moon-shaped dataset.



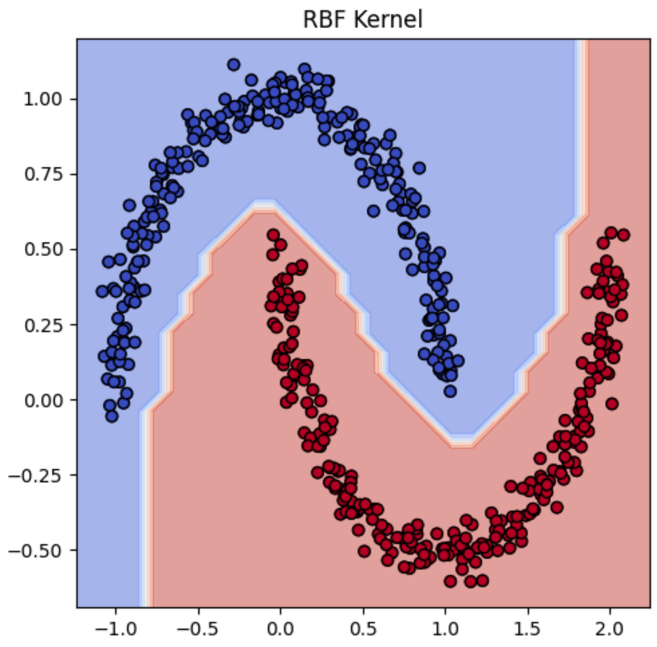
The decision boundary that was achieved by using a polynomial kernel to train a Support Vector Machine (SVM) on the artificial moon-shaped dataset is displayed in the graph.

The decision boundary of the polynomial kernel has a more intricate, curved shape than that of the linear kernel, and it more effectively divides the two non-linearly separable classes. The contour line, which represents the decision border, essentially encloses each class within its own region by following the basic moon-shaped structure of the data.

Still, it is clear that the polynomial kernel is still unable to precisely capture all of the fine details of the decision boundary. Even though it outperforms the linear kernel, some misclassifications persist, especially in the areas where the two moon-shaped clusters overlap or are closer together.

Finding a non-linear decision boundary is made possible by the polynomial kernel, which uses polynomial functions to transfer the input data into a higher-dimensional feature space. The decision boundary's complexity is determined by the polynomial kernel's degree; greater degrees may result in better separation but also raise the possibility of overfitting.

Although it might not be fully capturing the subtleties in the data, the polynomial kernel decision boundary in this instance seems to be a decent approximation of the true underlying decision boundary. Misclassifications are still evident in some data points, which are still on the incorrect side of the decision border.



The decision boundary that is produced when a Support Vector Machine (SVM) is trained using the radial basis function (RBF) kernel on the artificial moon-shaped dataset is depicted in the given graph.

The two interwoven moon-shaped classes have a complex shape and structure, which the RBF kernel's decision boundary closely matches. The decision boundary of the RBF kernel, in contrast to the linear and polynomial kernels, seems to accurately divide the two classes, with few misclassifications noted.

Radial basis functions are used by the RBF kernel to map the input data into an infinite-dimensional feature space, enabling the capture of extremely complicated and non-linear decision boundaries. The RBF kernel's decision boundary is curved and smooth, following the contours of the moon-shaped clusters to efficiently contain each class within its designated zone.

There don't seem to be many data items that are incorrectly labelled and fall outside of the decision boundary. However, rather than being a result of a kernel flaw, these misclassifications are probably caused by the overlap or intrinsic noise in the synthetic dataset.

The RBF kernel is a strong option for datasets with complex class distributions or non-linear structures because of its capacity to represent such complex decision boundaries. The RBF kernel can distinguish between classes that might not be linearly separable in the original feature space by efficiently transferring the data onto a higher-dimensional space.

The RBF kernel produces the best classification accuracy and fewest misclassifications because its decision boundary is the closest approximation to the actual underlying decision boundary of the moon-shaped dataset when compared to the linear and polynomial kernels.

TASK-2(C) :

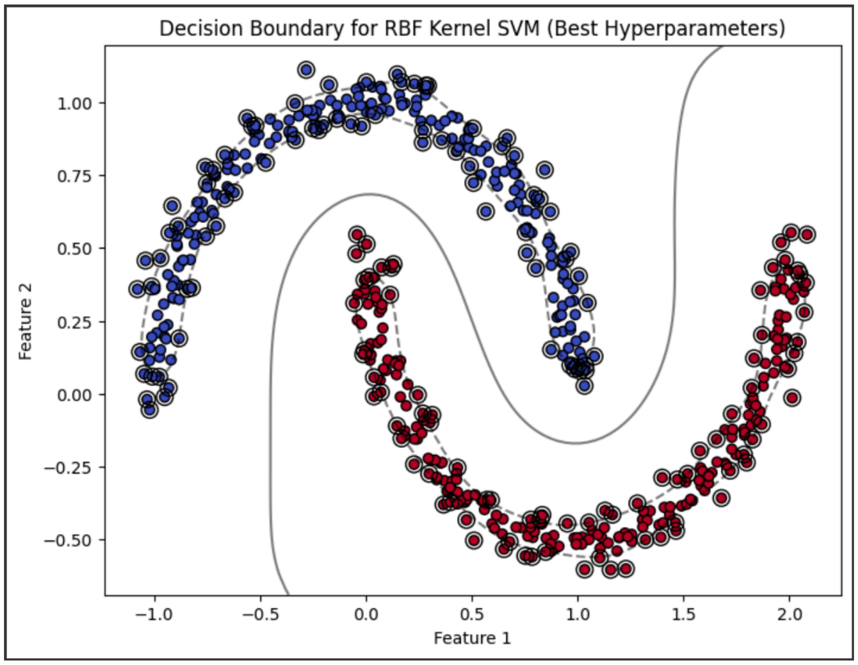
The code optimises the SVM model's performance using the RBF kernel by methodically experimenting with different combinations of the hyperparameters {C} and {gamma}. As the regularisation parameter, {C} controls the trade-off between minimising the classification error and maximising the margin. {C} and {gamma} yielded values of 0.1 and 10, respectively. The kernel coefficient in this case is defined by {gamma}, which affects the flexibility of the decision boundary. Based on the discovered values, it appears that the supplied dataset is best served by a moderate kernel coefficient ({gamma = 10}) and a somewhat low regularisation strength ({C = 0.1}). The model's increased prediction accuracy and capacity for generalisation as a result of fine-tuning these parameters demonstrate how well the hyperparameter tuning procedure works to improve SVM performance.

```
Best parameters: {'C': 0.1, 'gamma': 10}
```

TASK-2(D) :

The optimal hyperparameters for the RBF kernel SVM model were visualised by using the best values of gamma and C, which were found through a previous grid search method. Next, using these optimal hyperparameter values (best_gamma and best_C), an SVM model was constructed using the RBF kernel and fitted across the whole synthetic dataset (X and y).

Plotting the decision boundary required first constructing a mesh grid that included the dataset's two feature ranges. The decision values were then determined by evaluating the trained SVM model's decision_function on this mesh grid. The decision boundary was created by drawing the contour lines at levels 0 (solid line) and 1 (dashed line) of the decision values using the contour function.



The decision boundary is depicted in the final image as a smooth, oval-shaped contour that closely follows the two classes' underlying moon-shaped structure. The data samples for the two groups are shown by the red and blue points, respectively. With minimal misclassifications, the two non-linearly separable classes are effectively separated by the decision boundary reached by the RBF kernel SVM with optimal hyperparameters.

Plotted as hollow black circles are the support vectors, which are the training samples that are closest to the decision border and have a significant influence on defining its shape. The ideal C value balanced the trade-off between maximising the margin and minimising classification mistakes, while the best gamma value selection allowed the model to capture the complex decision boundary without overfitting.

This graphic amply illustrates how well the customised RBF kernel SVM models the intricate decision boundary of the synthetic dataset, obtaining good classification performance without experiencing problems with overfitting or underfitting.