PROGRAMMING ASSIGNMENT 5

Anushka Dadhich

B22CS097

Problem 1 - (Image Compression using K-means)

Task a:

computeCentroid() function, that takes n 3-dimensional features and returns their mean - It iterates over each dimension of the features, accumulates the sum, and then calculates the mean for each dimension to obtain the centroid.

Output -

```
(262144, 3)
Centroid: (137.3913345336914, 128.8587760925293, 113.11710739135742)
```

Task b:

mykmeans() function implements the K-means algorithm from scratch. It randomly selects initial centroids and then iteratively assigns data points to the nearest centroid and updates centroids until convergence. This process optimizes the arrangement of centroids to minimize intra-cluster variance, resulting in distinct clusters in the data.

Output -

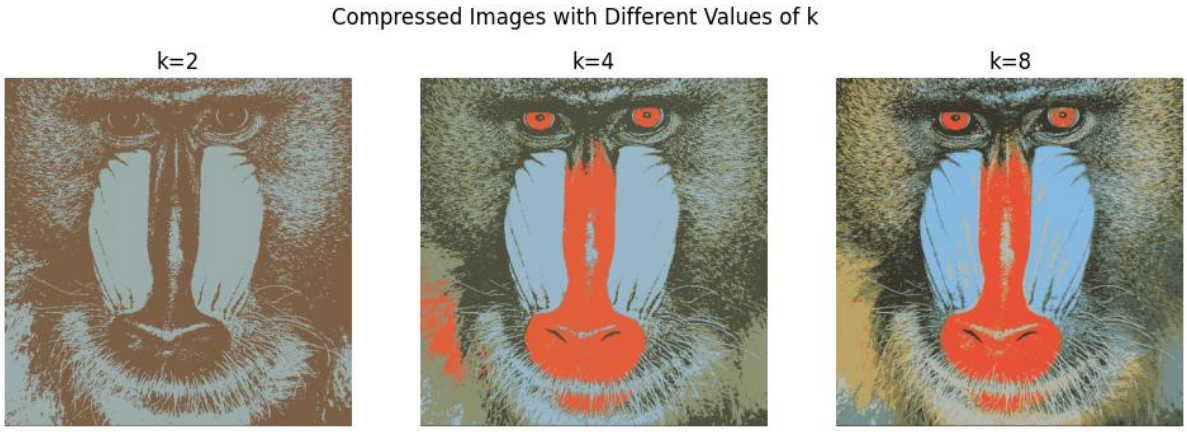
Just function created.

Task c:

The code compresses images using K-means clustering with varying values of k. This involves custom implementation. The mykmeans function assigns each pixel to the nearest centroid and reconstructs the compressed image accordingly. This process is repeated for different values of k, resulting in multiple compressed images.

The compress_image function assigns each pixel of the image to its nearest centroid and reconstructs the compressed image.

Output -

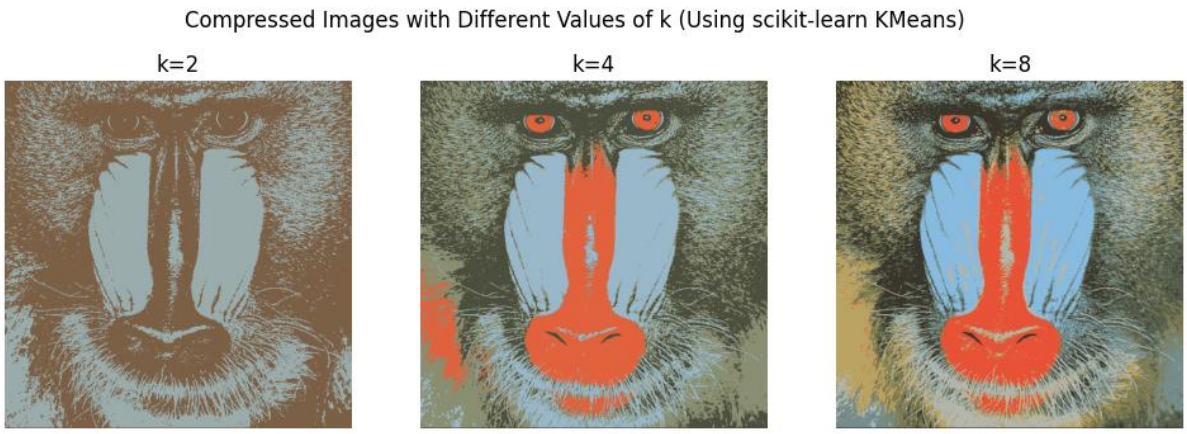


Task d:

The code compresses images for different values of k using scikits implementation of Kmeans.

Only slight differences have arisen due to initialization and convergence strategies, but overall, both implementations produce similar results.

Output -



Task e:

Idea:

The idea is to implement spatial coherence in image compression by modifying the clustering algorithm to consider both color similarity and spatial proximity. This approach ensures that pixels that are nearby in the original image are more likely to be assigned to the same cluster, preserving local structures and reducing artifacts like color bleeding or noise.

Implementation:

To implement this idea, we can modify the K-means clustering algorithm to incorporate spatial information.

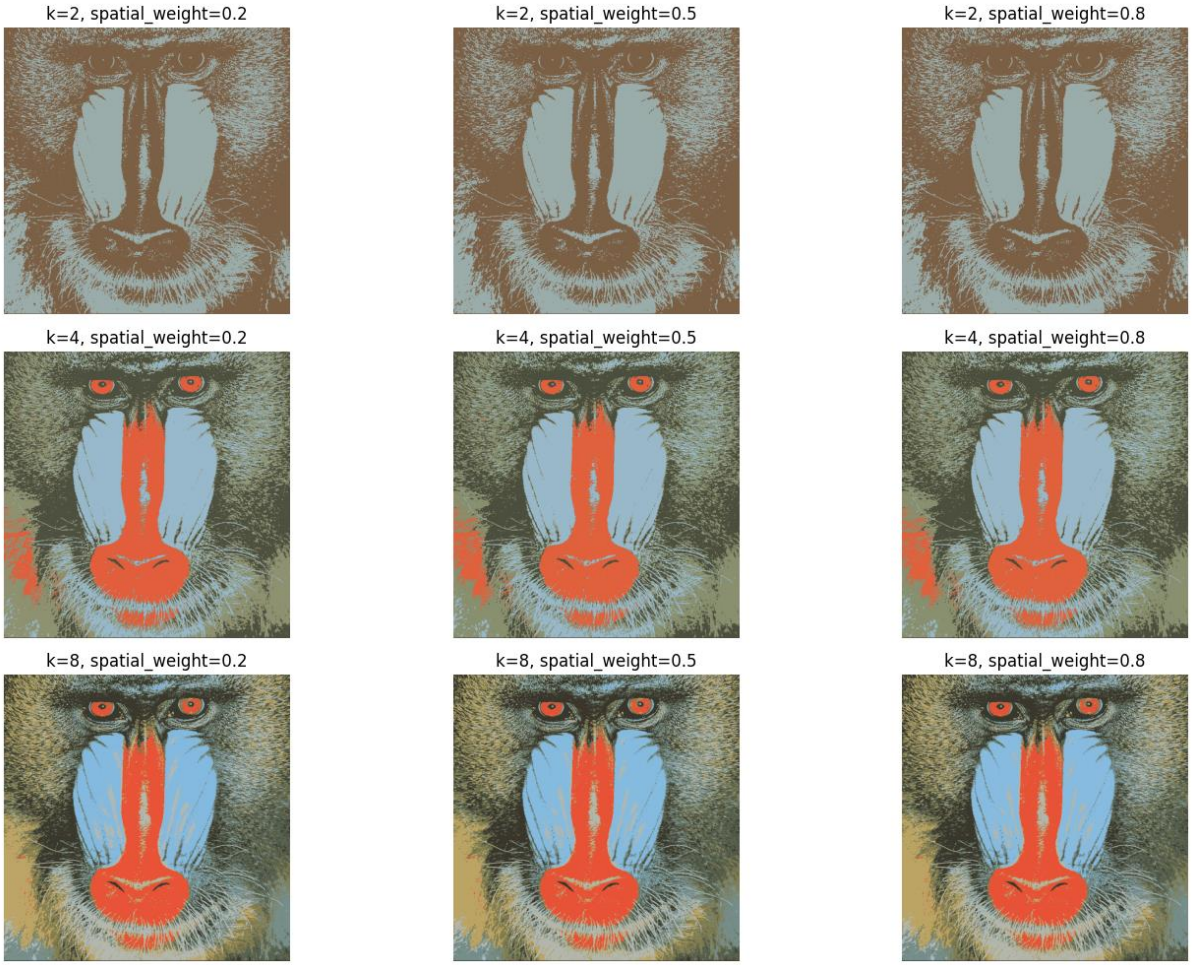
1. Flatten the input image into a 2D array of pixels.
2. Get the spatial coordinates of each pixel in the image.
3. Normalize the spatial coordinates to the range [0, 1].
4. Combine the color information and normalized spatial coordinates scaled by spatial_weight into a single feature vector for each pixel.
5. Perform K-means clustering on the combined feature vectors.
6. Reconstruct the compressed image based on the cluster centroids, preserving the original spatial arrangement of pixels.
7. Return the reconstructed image.

The spatial_weight parameter adjusts the importance of spatial proximity relative to color similarity in the clustering process. By multiplying the normalized spatial coordinates with spatial_weight, the algorithm can control the degree to which spatial information influences the clustering. Higher values of spatial_weight give more weight to spatial proximity, leading to stronger spatial coherence, while lower values prioritize color similarity.

Observations –

For some chosen good spatial_weights the image compression is enhanced.

Output –



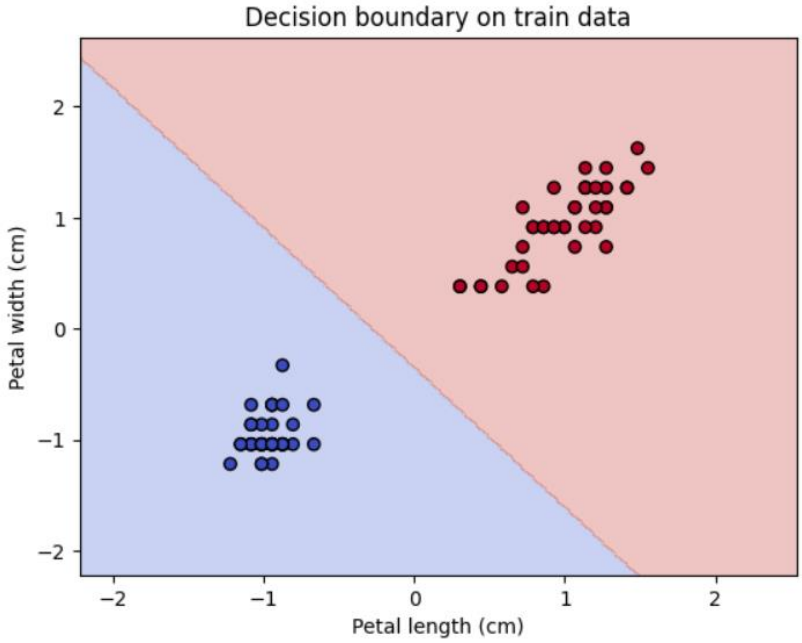
Problem 2 - (SVM)

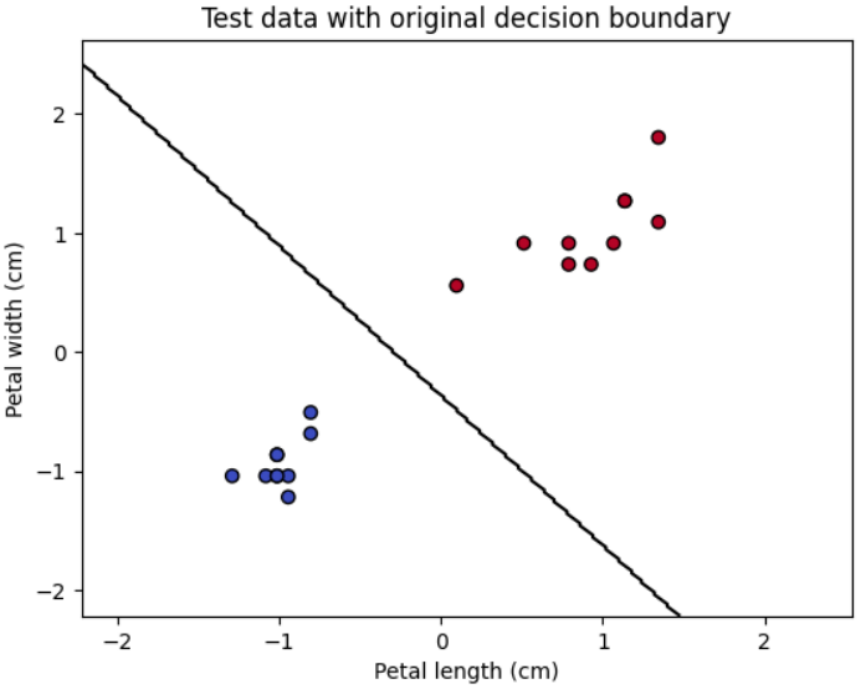
Task 1:

1. Loaded the Iris Dataset from sklearn, kept only 2 features from it and filtered the data to include only 'setosa' and 'versicolor' classes.
2. Normalized the features by (X_filtered - X_mean) / X_std and using the train_test_split(X_normalized, y_filtered, test_size=0.2, random_state=55) splitted into 80/20 split.
3. Training a Linear Support Vector Classifier (LinearSVC) using the training data (X_train and y_train) by defining the minimum and maximum values for features in the training data (x0_min, x0_max, x1_min, x1_max). A meshgrid (xx0, xx1) is created covering the range of feature values with a step size of 0.02.
4. Predictions are made for each point in the meshgrid using the trained classifier (Z). The decision boundary is plotted along with the training data using contourf to fill the regions with different colors based on the predicted classes and scatter to plot the training data points.
5. A scatterplot of the test data (X_test) is generated with the corresponding labels (y_test) and the original decision boundary from the training data is plotted using contour.
6. The plots are displayed to visualize the decision boundary on both the training and test data.

Observations - The decision boundary correctly classifies and fits both the test and training data.

Output:





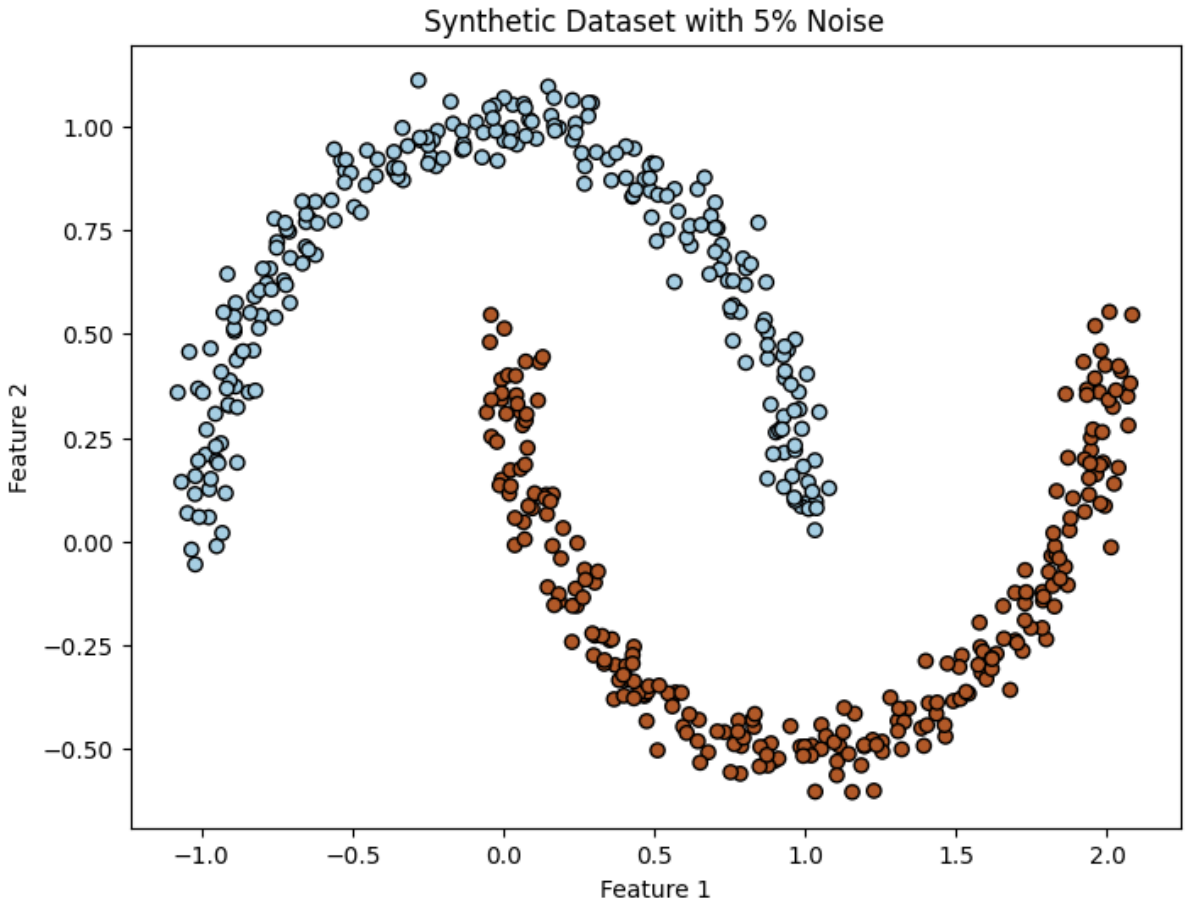
Task 2:

2.(a)

Generated dataset using make_moons of sklearn with 500 data points and 5% noice using make_moons(n_samples=500, noise=0.05).

Only added noise and no mis-classifications as said.

Output:



2.(b)

Implementing SVM using 3 Kernels:

- The Linear kernel is implemented using SVC with kernel='linear'.
- The Polynomial kernel is implemented using SVC with kernel='poly' and degree=3.
- The RBF kernel is implemented using SVC with kernel='rbf' and gamma='scale'.
- For each kernel, a subplot is created showing the dataset points and the decision boundary.

- Decision boundaries are obtained by using the decision_function method of each SVM model to generate predictions over a grid of points, which are then contoured to visualize the decision boundaries.
- Using the plot_decision_boundary() function to plot it, which generates a meshgrid covering the feature space, predicts class labels for each point, plots the decision boundary and data points, adjusts the plot's axes, and sets the title.

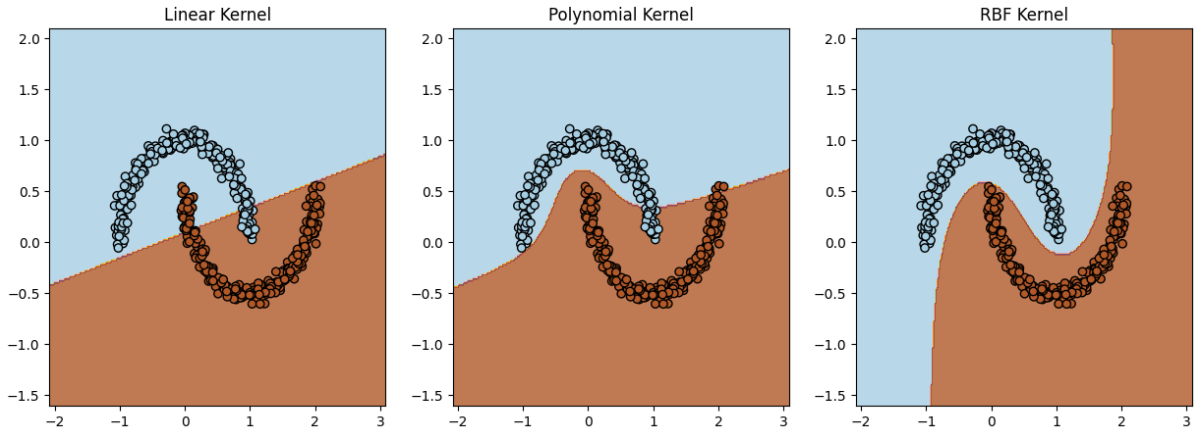
Analysis of Decision Boundaries:

- Linear Kernel: Produces linear decision boundaries. It separates the classes with a straight line.
- Polynomial Kernel: Produces more complex decision boundaries. With a degree of 3, it can capture nonlinear relationships between features.
- RBF Kernel: Produces highly flexible decision boundaries. It can adapt to complex patterns in the data by considering the similarity of points in the feature space.

Comments on Differences in Decision Boundaries:

- The Linear kernel produces the simplest decision boundary, suitable for linearly separable data.
- The Polynomial kernel introduces more complexity not completely fitting the data tough, allowing for non-linear decision boundaries.
- The RBF kernel provides the highest flexibility, capable of capturing intricate patterns in the data but may be prone to overfitting if not properly tuned.

Output:



2.(c)

Hyperparameter tuning is performed using grid search by the following steps:

- Define Parameter Grid - A parameter grid is defined with a range of C and gamma values. C controls the regularization strength, and gamma defines the influence of a single training example's distance.
- Initialize SVM Model - An SVM model with the RBF kernel is initialized.
- Perform Grid Search - Grid search is performed using GridSearchCV with 5-fold cross-validation which searches through the parameter grid to find the best combination of hyperparameters (C and gamma).
- The best combination is determined based on the highest cross-validated performance metric (default is mean accuracy).
- Then the best hyperparameters found by grid search are printed.

Output:

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

2(d):

The decision boundary of the best model is plotted using the plot_decision_boundary() function. This function visualizes the decision boundary on the training data.

Impact of C parameter:

C controls the regularization strength of the SVM model. A smaller C value indicates a softer margin, which allows for more misclassifications in the training data that can help prevent overfitting. Whereas a larger C value imposes a harder margin, leading to fewer misclassifications and lead overfitting if the model tries to fit the training data too closely.

Impact of gamma parameter:

gamma defines the influence of a single training example's distance. A smaller gamma value means a larger similarity radius, resulting in a smoother decision boundary which is less influenced by individual data points. Conversely, a larger gamma value makes the decision boundary more sensitive to variations in the training data, a more complex decision boundary that closely fits the training data, potentially leading to overfitting.

Hence,

Larger C values lead to a narrower margin and a more complex decision boundary, while smaller C values result in a wider margin and a simpler decision boundary.

Larger gamma values cause the decision boundary to closely fit the training data points, resulting in a more intricate boundary with high sensitivity to individual data points. Smaller gamma values, on the other hand, produce smoother decision boundaries that generalize better to unseen data.

Observations:

The decision boundary perfectly fits the points without mis-classifications.

Output: