PATTERN RECOGNITION AND MACHINE LEARNING
LAB REPORT
ASSIGNMENT-5 (COLLAB FILE)

Lavangi Parihar
B22EE044

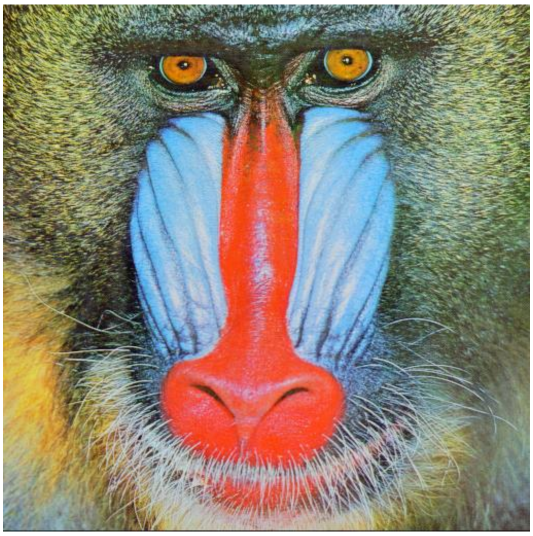
Question 1 Image Compression Using KMeans

a. Implementing computeCentroid:
- Extracts the pixel values from the image based on the provided indices.
- Calculates the mean of the RGB values across all the pixels.
- Returns the computed centroid as a numpy array.

b. Implementing myKmeans:
- Randomly selects `k` data points from `X` as initial cluster centers (centroids).
- For a maximum of `max_iters` iterations:
  Calculates the Euclidean distances between each data point and all cluster centers. Assigns each data point to the cluster with the nearest centroid.
  Computes new cluster centers by taking the mean of all data points assigned to each cluster.
  Checks if the centroids have converged (i.e., if they remain unchanged). If they have, the algorithm terminates.
- Returns the final cluster centers (centroids).

c. Compressing the Images:
Workflow
- Reshapes the image into a 2D array of pixels.
- Calculates distances from pixels to centroids.
- Assigns pixels to the nearest centroid.
- Replaces each pixel with the color of its nearest centroid.
- Reshapes the compressed pixels back into the original image shape.
- Saves the original and compressed images for different K values.

Original Image:



Results:

```
Shape of pixels array: (262144, 3)
```

d. Comparing results with kmeans implementation from sklearn library:

Workflow:

- Creates directories for saving compressed images.
- Saves the original image.
- Performs K-means clustering using scikit-learn's KMeans.
- Obtains centroids from the KMeans clustering.
- Compresses the image using obtained centroids.
- Saves the compressed images for different K values.

Result:

K	MyKMeans	KMeans Implementation from SKLearn
3		
4		

6		
8		

e. Spatial Coherence:

The code aims to compress images using K-means clustering with spatial coherence, where both color similarity and spatial proximity influence cluster assignments.

Functions Description:

- compute_spatial_distance(pixel1, pixel2): Computes the Euclidean distance between the coordinates of two pixels.
- mykmeans_spatial(X, k, max_iters=100, spatial_weight=0.5): Performs K-means clustering with spatial coherence using a weighted combination of color distance and spatial distance.
-save_compressed_images_spatial(image, k_values, save_path): Modifies the save_compressed_images function to use `mykmeans_spatial` for compression, producing spatially coherent compressed images.

Spatial Coherence in Compression:

- The spatial coherence in compression aims to preserve spatial relationships between pixels in addition to color similarity.

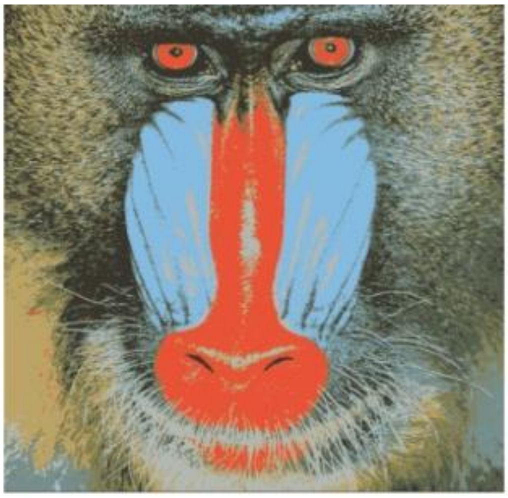
- By incorporating spatial distance into the clustering process, pixels that are nearby in the image are more likely to belong to the same cluster.

Result:

K=3

K=4

K=6



K=8

Question 2 Support Vector Machines

Task 1(a) Data Pre processing:

Data Preparation:
- Loading the Iris dataset using scikit-learn's datasets module with the `as_frame=True` parameter.
- Selecting only 'setosa' and 'versicolor' classes for binary classification.
- Extracting 'petal length' and 'petal width' features.

Data Normalization:
- Standardizing the dataset using StandardScaler to ensure features are on the same scale.
- Normalizing the features to have zero mean and unit variance.

Dataset Splitting:
- Splitting the normalized dataset into training and testing sets.
- Using a test size of 20% and a random state of 42 for reproducibility.

Results:

```
Shape of X_train: (80, 2)
Shape of X_test: (20, 2)
Shape of y_train: (80,)
Shape of y_test: (20,)
```

Task 1(b)
- Linear Support Vector Classifier (LinearSVC) is chosen for its effectiveness in binary classification tasks.
- LinearSVC is instantiated with a random state of 42 for reproducibility.
- The classifier is trained using the training data (X_train and y_train).
- The code generates decision boundaries for the trained Linear Support Vector Classifier (LinearSVC) on both training and testing data and saves the plots.

Function:
- `plot_decision_boundary_save(clf, X, y, title, save_path)`: Plots the decision boundary of a classifier along with the training data and saves the plot to a specified path.
  - Computes meshgrid to create a range of points for decision boundary plotting.
  - Predicts the class labels for the meshgrid points using the trained classifier.
  - Reshapes the predicted labels to match the meshgrid shape.
  - Plots the decision boundary using `contourf`.
  - Plots the training data points.

Results:


Decision Boundary on Training Data
2
1
0
−1
−2
Petal Width (cm)
−2
−1
0
1
2
Petal Length (cm)


Scatterplot of Test Data with Original Decision Boundary
2
1
0
−1
−2
Petal Width (cm)
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Petal Length (cm)

Task 2(a)
The code generates a synthetic dataset with two classes using the make_moons function from scikit-learn. It then introduces 5% noise to the dataset by flipping labels of a random subset of samples.

Dataset Generation:
- Synthetic dataset is generated using make_moons function with the following parameters:
  - Number of samples: 500.
  - Noise level: 0.05.
  - Random state: 42 for reproducibility.
- The dataset consists of two features and binary labels representing two classes.

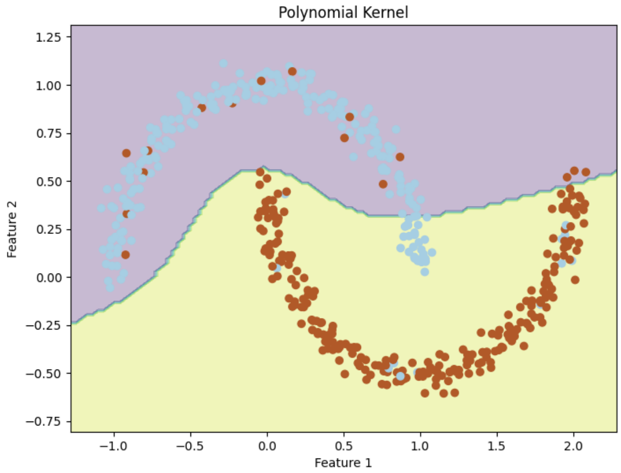
```
Shape of synthetic dataset: (500, 2)
Number of misclassifications: 500
```

Task 2(b)
The code aims to train Support Vector Machine (SVM) models with different kernels (Linear, Polynomial, and RBF) on a synthetic dataset and plot their decision boundaries.

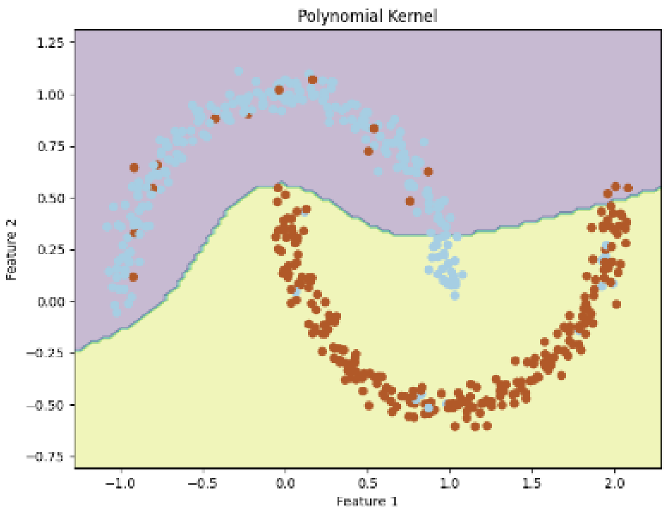
Model Training:
- SVM models with Linear, Polynomial, and RBF kernels are instantiated with specified parameters.
- Each model is fitted to the synthetic dataset (X_synthetic, y_synthetic) using the `fit` method.

Kernel Types:
- Linear Kernel: Assumes linear decision boundaries.
- Polynomial Kernel: Allows for nonlinear decision boundaries with a specified degree.
- RBF Kernel: Provides highly flexible decision boundaries, suitable for complex datasets.

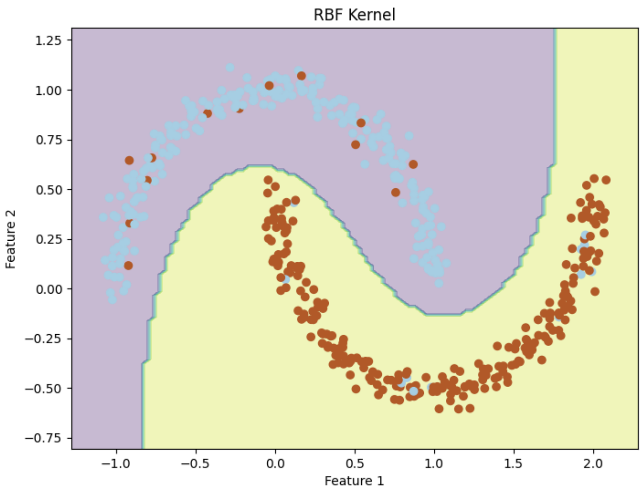
Analysis:
- The decision boundaries for each kernel type exhibit different characteristics:
  - Linear kernel: Straight lines separating the classes.
  - Polynomial kernel: More complex boundaries, capable of capturing nonlinear relationships.
  - RBF kernel: Highly flexible boundaries, adapting to the shape of the data clusters.
- The choice of kernel affects the model's ability to capture the underlying patterns in the data.
- Linear kernels may underperform on nonlinear datasets, while RBF kernels may overfit if not properly regularized.
- The choice of kernel should be based on the dataset's complexity and the desired balance between bias and variance.



LINEAR KERNEL



POLYNOMIAL KERNEL



RBF KERNEL

Task 2(c):
The code aims to perform hyperparameter tuning for the SVM model with the RBF kernel using grid search and find the best values of `C` and `gamma` for optimal model performance.

- A parameter grid is defined with different values of `C` and `gamma` to search over.
- `C` represents the regularization parameter, controlling the trade-off between margin size and classification error.
- `gamma` defines the influence of a single training example, affecting the flexibility of the decision boundary.
- GridSearchCV is instantiated with the SVM model with RBF kernel as the estimator and the defined parameter grid.
- 5-fold cross-validation is employed to evaluate each combination of hyperparameters.
- Accuracy is chosen as the scoring metric to measure model performance.
- The best SVM model with the optimal hyperparameters is retrieved using `grid_search.best_estimator_`.

Result:

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Best hyperparameters: {'C': 1, 'gamma': 1}
```

Task 2(d):
The function `plot_decision_boundary_save` aims to plot the decision boundary of a machine learning model on a 2D dataset and save the plot as an image.

Function Description:
- Inputs:
  - `model`: The trained machine learning model used to predict class labels.
  - `X`: The feature matrix of the dataset containing two features.
  - `y`: The target labels of the dataset.
  - `title`: The title of the plot.
  - `save_path`: The file path where the plot will be saved.
- Process:
  - Computes the minimum and maximum values of each feature dimension to define the plot boundaries.
  - Creates a meshgrid of points spanning the feature space.
  - Predicts the class labels for each point in the meshgrid using the provided model.
  - Reshapes the predicted labels to match the meshgrid shape.
  - Plots the decision boundary using `contourf` to visualize the decision regions.
  - Plots the data points using `scatter` to show the distribution of the dataset.
  - Adds axis labels and a title to the plot.
  - Saves the plot as an image at the specified `save_path`.
  - Closes the plot to release memory resources.

Plotting Decisions:
- The function generates a filled contour plot to represent the decision boundary, allowing for clear visualization of decision regions.
- Data points are overlaid on the plot to show the distribution of the dataset and how it relates to the decision boundary.

- Axis labels and a title are included to provide context and clarity to the plot.

Result: