Instructions:

- You need to code in this jupyter notebook only.
- Download this notebook and import in your jupyter lab.
- You need to write a partial code for step 0 to step 8 mentioned with prefix ##
- Fill the blanks where it is instructed in comments.
- Leave other codes, structure as it is.
- Follow all the instructions commented in the cells.

Answer the questions given at the end of this notebook within your report.

Upload this jupyter notebook after completion with your partial code and the report in one file in PDF format. Your file name should be yourname_lab4.pdf

Also upload the resulting image showing all the selected points and boundary line between them after LDA analysis.

Your submission should contain the pdf file and the output plot. Upload it on the LMS before the due time.

```
import numpy as np
import cv2
import matplotlib
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
matplotlib.use('TkAgg')

##-----------------------------------------------------
## Step 0: Install all other dependencies that occur at run time if
any module not found.
##-----------------------------------------------------

Number_of_points = 25  ## Number of points you want select from each
strip. Recommended >= 20

img = cv2.imread('Indian_Flag.jpg') ## Read the given image

def select_points(img, title):
    fig, ax = plt.subplots()
    #-------------------------------------------
    ## step 1: Convert the img from BGR to RGB using cv2 and display
it using cv2.imshow
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #converting BGR to
RGB (since matplotlib reads images in RGB format)
    ax.imshow(img_rgb)
    ## step 2: Put title of the image
    ax.set_title(title)
    ##-----------------------------------------

    # Set the cursor style to a plus sign
```

```
    fig.canvas.manager.set_window_title('Select Points')
    cursor = matplotlib.widgets.Cursor(ax, useblit=True, color='red',
linewidth=1)
    plt.show(block=False)  # Show the image without blocking

    k = 0
    points = [] ## Create here an empty list to store points

    while k < Number_of_points:
        xy = plt.ginput(1, timeout=0)  # Non-blocking input
        if len(xy) > 0:
            col, row = map(int, xy[0])  # Convert to integer
            ##-------------------------------------------------
            ## Step 3: Collect RGB values at the clicked positions
(col, row) and print it.
            rgb = img_rgb[row, col]
            print(f"Point {k+1}: Row={row}, Col={col}, RGB={rgb}")
            ##-------------------------------------------------

            k += 1
            points.append([row, col, img[row, col]])  # Store RGB
values in empty list points.

            # Display colored dot on the image
            plt.scatter(col, row, c='black', marker='o', s=10)

            # Redraw the image to include the dot
            plt.draw()

    plt.close()  # Close the window after all points are collected
    return points ## Fill this blank

##-------------------------------------------------------------------
## Step4: fill the blanks for Selected points from saffron strip
pts_saffron = select_points(img, 'Select points from Saffron Strip')
## Step5: fill the blanks for Selected points from white strip)
pts_white = select_points(img, 'Select points from White Strip')
## Step6: fill the blanks for Selected points from green strip
pts_green = select_points(img, 'Select points from Green Strip')
##-------------------------------------------------------------------
```

```
Point 1: Row=268, Col=539, RGB=[250  95  28]
Point 2: Row=282, Col=539, RGB=[241  84  15]
Point 3: Row=294, Col=543, RGB=[234  85  17]
Point 4: Row=287, Col=560, RGB=[244  87  16]
Point 5: Row=282, Col=548, RGB=[243  89  19]
Point 6: Row=273, Col=551, RGB=[247  93  23]
Point 7: Row=275, Col=563, RGB=[242  88  16]
Point 8: Row=297, Col=563, RGB=[246  89  20]
Point 9: Row=297, Col=577, RGB=[246  89  18]
```

```
Point 10: Row=285, Col=575, RGB=[249  88  18]
Point 11: Row=277, Col=575, RGB=[248  87  17]
Point 12: Row=275, Col=592, RGB=[243  84  16]
Point 13: Row=292, Col=592, RGB=[245  83  11]
Point 14: Row=282, Col=592, RGB=[250  87  18]
Point 15: Row=270, Col=604, RGB=[246  84  22]
Point 16: Row=285, Col=609, RGB=[246  80   6]
Point 17: Row=299, Col=609, RGB=[239  81   7]
Point 18: Row=299, Col=621, RGB=[238  80   6]
Point 19: Row=290, Col=626, RGB=[247  84  17]
Point 20: Row=270, Col=618, RGB=[246  81  25]
Point 21: Row=268, Col=630, RGB=[243  85  12]
Point 22: Row=265, Col=635, RGB=[247  69   5]
Point 23: Row=265, Col=647, RGB=[240  79   9]
Point 24: Row=277, Col=635, RGB=[240  73   3]
Point 25: Row=290, Col=638, RGB=[240  84   9]
Point 1: Row=316, Col=534, RGB=[220 216 239]
Point 2: Row=319, Col=548, RGB=[230 221 242]
Point 3: Row=311, Col=531, RGB=[228 225 218]
Point 4: Row=326, Col=541, RGB=[229 226 235]
Point 5: Row=328, Col=563, RGB=[237 236 242]
Point 6: Row=316, Col=563, RGB=[231 235 236]
Point 7: Row=311, Col=633, RGB=[220 221 223]
Point 8: Row=309, Col=650, RGB=[229 226 243]
Point 9: Row=306, Col=659, RGB=[220 218 232]
Point 10: Row=319, Col=659, RGB=[214 211 230]
Point 11: Row=331, Col=657, RGB=[217 218 236]
Point 12: Row=331, Col=635, RGB=[222 224 239]
Point 13: Row=321, Col=638, RGB=[229 226 245]
Point 14: Row=321, Col=645, RGB=[225 224 242]
Point 15: Row=331, Col=548, RGB=[229 231 244]
Point 16: Row=331, Col=558, RGB=[229 232 241]
Point 17: Row=309, Col=536, RGB=[223 221 232]
Point 18: Row=326, Col=524, RGB=[219 219 229]
Point 19: Row=331, Col=570, RGB=[217 217 225]
Point 20: Row=316, Col=572, RGB=[228 226 231]
Point 21: Row=323, Col=626, RGB=[216 214 225]
Point 22: Row=331, Col=640, RGB=[230 228 249]
Point 23: Row=311, Col=628, RGB=[219 219 217]
Point 24: Row=333, Col=628, RGB=[219 221 234]
Point 25: Row=311, Col=548, RGB=[234 224 248]
Point 1: Row=345, Col=524, RGB=[30 99 71]
Point 2: Row=350, Col=524, RGB=[39 96 77]
Point 3: Row=362, Col=524, RGB=[34 98 72]
Point 4: Row=365, Col=543, RGB=[29 99 71]
Point 5: Row=355, Col=539, RGB=[28 98 70]
Point 6: Row=352, Col=551, RGB=[ 29 102  75]
Point 7: Row=350, Col=541, RGB=[ 30 102  78]
Point 8: Row=367, Col=539, RGB=[27 97 69]
```

```
Point 9: Row=367, Col=558, RGB=[33 98 74]
Point 10: Row=355, Col=563, RGB=[ 29 102  75]
Point 11: Row=355, Col=577, RGB=[29 99 73]
Point 12: Row=360, Col=577, RGB=[ 29 100  70]
Point 13: Row=374, Col=577, RGB=[29 96 65]
Point 14: Row=372, Col=587, RGB=[32 97 67]
Point 15: Row=365, Col=592, RGB=[26 93 62]
Point 16: Row=355, Col=599, RGB=[32 98 70]
Point 17: Row=355, Col=616, RGB=[22 88 61]
Point 18: Row=360, Col=616, RGB=[31 97 70]
Point 19: Row=369, Col=611, RGB=[24 93 65]
Point 20: Row=369, Col=638, RGB=[25 87 62]
Point 21: Row=365, Col=635, RGB=[27 91 65]
Point 22: Row=355, Col=638, RGB=[22 86 60]
Point 23: Row=360, Col=657, RGB=[20 82 57]
Point 24: Row=355, Col=655, RGB=[25 89 63]
Point 25: Row=365, Col=626, RGB=[25 89 63]

# Convert RGB values to Lab color space
def rgb_to_lab(rgb):
    return cv2.cvtColor(np.uint8([[rgb]]), cv2.COLOR_RGB2Lab)[0][0]

saffron_lab = np.array([rgb_to_lab(rgb) for _, _, rgb in pts_saffron])
white_lab = np.array([rgb_to_lab(rgb) for _, _, rgb in pts_white])
green_lab = np.array([rgb_to_lab(rgb) for _, _, rgb in pts_green])

## Step7: Extract a* and b* components from Lab color space
a_features = np.hstack((saffron_lab[:, 1], white_lab[:, 1],
green_lab[:, 1]))
b_features = np.hstack((saffron_lab[:, 2], white_lab[:, 2],
green_lab[:, 2]))

# Map class labels to numeric values
class_mapping = {'Saffron': 0, 'White': 1, 'Green': 2}
y = np.array([class_mapping[label] for label in ['Saffron'] *
Number_of_points + ['White'] * Number_of_points + ['Green'] *
Number_of_points])

plt.figure()
plt.scatter(a_features[:Number_of_points],
b_features[:Number_of_points], c='b', marker='o', s=50,
label='Saffron')
plt.scatter(a_features[Number_of_points:2*Number_of_points],
b_features[Number_of_points:2*Number_of_points], c='g', marker='^',
s=50, label='White')
plt.scatter(a_features[2*Number_of_points:],
b_features[2*Number_of_points:], c='r', marker='*', s=50,
label='Green')
plt.legend(['Saffron', 'White', 'Green'], loc='best')
plt.xlabel('a* component (Green-Red)')  ## Provide x label
```

```python
plt.ylabel('b* component (Blue-Yellow)') ## Provide y label
plt.title('Feature Space: a* vs b* components') ## Provide title
plt.grid()
plt.show()

##------------------------------------------------------------
# Step 8: Perform LDA analysis using LinearDiscriminantAnalysis() and
lda.fit()
X = np.column_stack((a_features, b_features))
lda = LinearDiscriminantAnalysis()
lda.fit(X, y)
##-----------------------------------------------------------
```

```
LinearDiscriminantAnalysis()
```

```python
# Plot LDA boundaries
plt.figure()
plt.scatter(a_features[:Number_of_points],
b_features[:Number_of_points], c='b', marker='o', s=50,
label='Saffron')
plt.scatter(a_features[Number_of_points:2*Number_of_points],
b_features[Number_of_points:2*Number_of_points], c='g', marker='^',
s=50, label='White')
plt.scatter(a_features[2*Number_of_points:],
b_features[2*Number_of_points:], c='r', marker='*', s=50,
label='Green')

plt.xlabel('a* component (Green-Red)')  ## Provide x label
plt.ylabel('b* component (Blue-Yellow)') ## Provide y label
plt.title('LDA boundaries (linear model) for Colors of the Indian
Flag')

# Plot the decision boundaries
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 100),
np.linspace(ylim[0], ylim[1], 100))
Z = lda.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, colors='k', linewidths=2, linestyles='solid')
plt.legend(loc='best')
plt.grid()
plt.show()
```

Report:

Answer the following questions within your report:

1. What are the key assumptions underlying LDA, and how do these assumptions influence the model's performance?

- LDA assumes Gaussian distribution for each class and equal covariance matrices across all classes.
- Violating these assumptions leads to inaccurate decision boundaries and suboptimal classification performance.

2. What are the hyperparameters in LDA, and how do they affect the outcome of the model?

- **solver:** Choice of algorithm ('svd', 'lsqr', 'eigen') affects computational speed and numerical stability.
- **shrinkage:** Regularization parameter that reduces overfitting by adjusting covariance estimation.

3. What methods can be used to assess an LDA model's effectiveness?

- Classification accuracy and confusion matrix to evaluate prediction performance.
- Cross-validation and ROC-AUC score for robust performance estimation.

4. What are some common challenges or limitations associated with LDA, and how can they be addressed or mitigated?

- Small sample sizes and non-linear boundaries reduce LDA effectiveness. Use regularized LDA or switch to kernel methods.
- Outliers distort decision boundaries. Remove outliers or use robust covariance estimation.

5. What practical applications does this assignment have in real-world situations, and what benefits does it offer in those specific scenarios?

- Quality control in manufacturing and medical imaging for disease diagnosis using color-based classification.
- Benefits include real-time automated classification, reduced manual effort, and consistent objective decisions.