```
#ABC

import numpy as np

# Define your hyperspectral dataset here as 'hyperspectral_data'
# Define the number of food sources 'N' and maximum iterations 'g' here
# Define other necessary functions as mentioned in the algorithm

def compute_fitness(food_source):
    # Define your fitness function here
    pass

def explore_neighbor(food_source):
    # Define how to explore a neighboring food source
    pass

def generate_new_food_source():
    # Define how to generate a new food source
    pass

# Step 1: Partition bands into subspaces using ISD method
# Implement this step based on your dataset and method

# Step 2: Initialize t
t = 15

# Step 3: Initialize population A with random band subsets
population_size = N  # Number of food sources
subspace_band_count = k  # Number of bands to select from each subspace
population = []

for i in range(population_size):
    food_source = np.random.choice(hyperspectral_data,
size=subspace_band_count, replace=False)
    population.append(food_source)

# Step 4: While the termination criterion is not met (t < g)
while t < g:
    # Step 5: Compute fitness of each food source
    fitness_values = [compute_fitness(food_source) for food_source in
population]

    # Step 5 (cont): Sort the food sources by fitness
    sorted_indices = np.argsort(fitness_values)[::-1]
    population = [population[i] for i in sorted_indices]
    fitness_values.sort(reverse=True)

    # Step 6: Update employed bee population
```

```
    employed_bees = population[:population_size // 2]
    new_employed_bees = []

    for food_source in employed_bees:
        neighbor_food_source = explore_neighbor(food_source)
        if compute_fitness(neighbor_food_source) >
compute_fitness(food_source):
            new_employed_bees.append(neighbor_food_source)
        else:
            new_employed_bees.append(food_source)

    # Step 7: Update onlooker population
    onlookers = population[population_size // 2:]
    new_onlookers = []

    for food_source in onlookers:
        neighbor_food_source = explore_neighbor(food_source)
        if compute_fitness(neighbor_food_source) >
compute_fitness(food_source):
            new_onlookers.append(neighbor_food_source)
        else:
            new_onlookers.append(food_source)

    # Step 8: Combine employed bee and onlooker populations
    population = new_employed_bees + new_onlookers

    # Step 9: Abandon and search for new food sources
    abandoned_food_source = population[-1]
    new_food_source = generate_new_food_source()

    if compute_fitness(new_food_source) >
compute_fitness(abandoned_food_source):
        population[-1] = new_food_source

    # Step 10: Increment t
    t += 1

# Step 12: Output the optimal band combination
optimal_band_combination = population[0]
```

BAND DECOMPOSITION

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from google.colab import files
import io  # Import the 'io' module
```

```
# Upload your 2D PNG images (e.g., 'image1.png', 'image2.png', ...)
using the files.upload() function
uploaded = files.upload()

# Initialize an empty list to store the image data
image_data_list = []

# Loop through the uploaded files
for file_name, file_content in uploaded.items():
    # Read each PNG image using matplotlib
    image_data = plt.imread(io.BytesIO(file_content))  # Use io.BytesIO
to read from file_content

    # Ensure the image has 2 dimensions (height, width) even if there's
an extra channel dimension
    if len(image_data.shape) > 2:
        image_data = image_data[:, :, 0]  # Take only the first channel
(grayscale)

    # Append the image data to the list
    image_data_list.append(image_data)

# Convert the list of image data into a NumPy array
image_data_array = np.stack(image_data_list)

# Get the number of samples (images), height, and width of each image
num_samples, height, width = image_data_array.shape

# Reshape the data into a 2D array (samples x pixels)
data_2d = image_data_array.reshape((num_samples, height * width))

# Determine the maximum number of components based on available
features (pixels)
max_components = min(num_samples, height * width)

# Perform PCA-based dimensionality reduction
num_components = min(10, max_components)  # Number of components to
retain (limited by available features)
pca = PCA(n_components=num_components)
reduced_data = pca.fit_transform(data_2d)

# Reconstruct the data with reduced dimensionality
reconstructed_data = pca.inverse_transform(reduced_data)

# Reshape the reconstructed data to match the original image shape
reconstructed_images = reconstructed_data.reshape((num_samples, height,
width))
```

```
# Visualize the original and reconstructed images for a selected sample
selected_sample = 0

# Plot the original image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_data_array[selected_sample], cmap='gray')
plt.title(f'Original Image {selected_sample + 1}')

# Plot the reconstructed image
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_images[selected_sample], cmap='gray')
plt.title(f'Reconstructed Image {selected_sample + 1}')

plt.tight_layout()
plt.show()
```

BAND SUBSET DECOMPOSITION

```
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
import io

# Upload your PNG images and process them as you did before
uploaded = files.upload()
image_data_list = []

for file_name, file_content in uploaded.items():
    image_data = plt.imread(io.BytesIO(file_content))
    if len(image_data.shape) > 2:
        image_data = image_data[:, :, 0]
    image_data_list.append(image_data)

image_data_array = np.stack(image_data_list)

# Reshape the image data into a 2D array (samples x pixels)
num_samples, height, width = image_data_array.shape
data_2d = image_data_array.reshape((num_samples, height * width))

# Calculate the correlation matrix between images (samples)
correlation_matrix = np.corrcoef(data_2d, rowvar=False)

num_bands = image_data_array.shape[-1]

# Create an array of band numbers from 1 to num_bands
band_numbers = np.arange(1, num_bands + 1)
```

```
# Calculate the covariance coefficients for each band
covariance_coefficients = np.diag(correlation_matrix)

# Plot the covariance coefficients against band numbers
plt.figure(figsize=(10, 6))
plt.plot(band_numbers, covariance_coefficients, marker='o',
linestyle='-')
plt.xlabel('Band Numbers')
plt.ylabel('Covariance Coefficients')
plt.title('Covariance Coefficients vs Band Numbers')
plt.grid(True)
plt.show()
```

CONVERTING .MAT FILE INTO NUMPY ARRAY

```
import scipy.io
import numpy as np
from google.colab import files
from scipy.stats import pearsonr

# Upload the .mat file
uploaded = files.upload()

# Load the uploaded .mat file
mat_file = scipy.io.loadmat(list(uploaded.keys())[0])

# Access the data variable in the .mat file
data = mat_file['indian_pines_corrected']

# Reshape the data to have columns as length*width and rows as height
height, length, width = data.shape

# Reshape the data into a 2D NumPy array
data_2d = data.reshape((height, length * width))

# Now, 'data_2d' contains your data as a 2D NumPy array with columns as
length*width and rows as height

correlation_matrix = np.corrcoef(data_2d, rowvar=False)

adjacent_band_correlations = []

for i in range(correlation_matrix.shape[0] - 1):
    corr_coefficient, _ = pearsonr(correlation_matrix[i],
correlation_matrix[i + 1])
    adjacent_band_correlations.append(corr_coefficient)

# Plot the correlation coefficients for adjacent bands
```

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adjacent_band_correlations) + 1),
adjacent_band_correlations, marker='o', linestyle='-')
plt.xlabel('Band Pairs')
plt.ylabel('Correlation Coefficient')
plt.title('Correlation Coefficients for Adjacent Bands')
plt.grid(True)
plt.show()
```

```
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
from scipy.stats import pearsonr
from scipy.stats import entropy
# Upload the .mat file
uploaded = files.upload()

# Load the uploaded .mat file
mat_file = scipy.io.loadmat(list(uploaded.keys())[0])

# Access the data variable in the .mat file
data = mat_file['indian_pines_corrected']

# Reshape the data to have columns as length*width and rows as height
length, width, height = data.shape
print(length)
print(width)
print(height)
# Reshape the data into a 2D NumPy array
data_2d = data.reshape(( length * width , height))
num_columns = data_2d.shape[1]
print(num_columns)
# Now, 'data_2d' contains your data as a 2D NumPy array with columns as
length*width and rows as height
correlation_coefficients = []

# Calculate correlation coefficients between adjacent bands
for i in range(num_columns - 1):
    column1 = data_2d[:, i]
    column2 = data_2d[:, i + 1]
    correlation_coefficient = np.corrcoef(column1, column2)[0, 1]
    correlation_coefficients.append(correlation_coefficient)

print(correlation_coefficients)

#Normalization
min_corr = min(correlation_coefficients)
```

```
max_corr = max(correlation_coefficients)
normalized_correlation_coefficients = [(x - min_corr) / (max_corr -
min_corr) for x in correlation_coefficients]

vsize = len(correlation_coefficients)
print(vsize)

#entropy calcuation
band_entropies = [entropy(data_2d[:, i], base=2) for i in
range(num_columns)]
print(band_entropies)
print(len(band_entropies))

#plotting the graph
plt.plot(normalized_correlation_coefficients)
plt.show()
```

```
import scipy.io
import numpy as np
from google.colab import files
# Upload the .mat file
uploaded = files.upload()

# Load the uploaded .mat file
mat_file = scipy.io.loadmat(list(uploaded.keys())[0])

# Access the data variable in the .mat file
data = mat_file['indian_pines_gt']
print(data.shape)
# Reshape the data to have columns as length*width and rows as height
length, width = data.shape
print(length)
print(width)

D1 = []
D2 = []
D3 = []
D4 = []
D5 = []
D6 = []
D7 = []
D8 = []
D9 = []
D10 = []
D11 = []
D12 = []
D13 = []
D14 = []
```

```
D15 = []
D16 = []


for i in range(length - 1):
  for j in range(width - 1):
      if data[i][j] == 1 :
        D1.append(data[i][j])

      if data[i][j] == 2 :
        D2.append(data[i][j])

      if data[i][j] == 3 :
        D3.append(data[i][j])

      if data[i][j] == 4 :
        D4.append(data[i][j])

      if data[i][j] == 5 :
        D5.append(data[i][j])

      if data[i][j] == 6 :
        D6.append(data[i][j])

      if data[i][j] == 7 :
        D7.append(data[i][j])

      if data[i][j] == 8 :
        D8.append(data[i][j])

      if data[i][j] == 9 :
        D9.append(data[i][j])

      if data[i][j] == 10 :
        D10.append(data[i][j])

      if data[i][j] == 11 :
        D11.append(data[i][j])

      if data[i][j] == 12 :
        D12.append(data[i][j])

      if data[i][j] == 13 :
        D13.append(data[i][j])

      if data[i][j] == 14 :
        D14.append(data[i][j])

      if data[i][j] == 15 :
```

```
        D15.append(data[i][j])

      if data[i][j] == 16 :
        D16.append(data[i][j])

print(len(D1))
print(len(D2))
print(len(D3))
print(len(D4))
print(len(D5))
print(len(D6))
print(len(D7))
print(len(D8))
print(len(D9))
print(len(D10))
print(len(D11))
print(len(D12))
print(len(D13))
print(len(D14))
print(len(D15))
print(len(D16))
```

```
from google.colab import files
import numpy as np
import scipy.io
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
import numpy as np
import random
import scipy.io
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

g = 10
N = 30
p = 5
breakpoints = [0, 36, 61, 75, 103, 144, 200]
```

```
mat_data = scipy.io.loadmat('Indian_pines_corrected.mat')
data_3d = mat_data['indian_pines_corrected']
length, width, height = data_3d.shape
new_height = height + 1
new_data = np.zeros((length, width, new_height))

for i in range(length):
    for j in range(width):
        for k in range(height):
            new_data[i][j][k] = data_3d[i][j][k]

k = 7
mat_data = scipy.io.loadmat('Indian_pines_gt.mat')
gtdata = mat_data['indian_pines_gt']

for i in range(len(gtdata)):
    for j in range(len(gtdata[0])):
        new_data[i][j][height] = gtdata[i][j]

newdata_2d = new_data.reshape((length * width, new_height))
data = newdata_2d

def classification_error(food_source):
    X = data[:, food_source]
    y = data[:, -1]
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42 , stratify=y)
    model = DecisionTreeClassifier()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    classification_error = 1 - accuracy
    return classification_error


def compute_fitness(food_source):
    return 1-classification_error(food_source)

# def generate_new_food_source():
#   food_source = []
#   food_source = random.sample(range(0,200),k*6)

#   return food_source


def generate_new_food_source():
    food_source = []
    for i in range(len(breakpoints) - 1):
        start = breakpoints[i]
```

```
        end = breakpoints[i + 1]

        elements_from_range = random.sample(range(start, end),k)
        food_source.extend(elements_from_range)

    return food_source

def calculate_probabilities(fitness_values):
    total_fitness = sum(fitness_values)
    probabilities = [fitness / total_fitness for fitness in
fitness_values]
    return probabilities

def roulette_wheel_selection(elements, probabilities):
    if len(elements) != len(probabilities):
        raise ValueError("The number of elements must match the number
of probabilities")

    total_probability = sum(probabilities)
    if total_probability != 1.0:
        raise ValueError("Probabilities must sum to 1.0")

    r = random.uniform(0, 1)
    cumulative_prob = 0

    for element, probability in zip(elements, probabilities):
        cumulative_prob += probability
        if r <= cumulative_prob:
            return element

# Step 1: Partition bands into subspaces using ISD method
# Implement this step based on your dataset and method

# Step 2: Initialize t
t = 15

# Step 3: Initialize population A with random band subsets
population_size = N  # Number of food sources
subspace_band_count = k*6  # Number of bands to select from each
subspace
population = []
improvement = []
for i in range(population_size):
    population.append(generate_new_food_source())
    improvement.append(0)

# Step 4: While the termination criterion is not met (t < g)
while t < g:
```

```
    # Step 5: Compute fitness of each food source
    fitness_values = [compute_fitness(food_source) for food_source in
population]

    # Step 5 (cont): Sort the food sources by fitness
    sorted_indices = np.argsort(fitness_values)[::-1]
    population = [population[i] for i in sorted_indices]
    fitness_values.sort(reverse=False)

    # Step 6: Update employed bee population
    employed_bees = population[:population_size // 2]
    new_employed_bees = []

    i=0
    for food_source in employed_bees:
        neighbor_food_source = random.choice(employed_bees)
        b = random.randint(subspace_band_count)
        new_food_source = food_source
        new_food_source[b]= neighbor_food_source[b]
        if compute_fitness(new_food_source) >
compute_fitness(food_source):
            new_employed_bees.append(new_food_source)
        else:
            new_employed_bees.append(food_source)
            improvement[i]+=1;
        i += 1

    # Step 7: Update onlooker population
    onlookers = population[population_size // 2:]
    new_onlookers = []
    probabilities =
calculate_probabilities(fitness_values[:population_size // 2])
    for food_source in onlookers:
        neighbor_food_source = roulette_wheel_selection(employed_bees,
probabilities)
        new_food_source = food_source
        new_food_source[b]= neighbor_food_source[b]
        if compute_fitness(new_food_source) >
compute_fitness(food_source):
            new_onlookers.append(new_food_source)
        else:
            new_onlookers.append(food_source)
            improvement[i]+=1;
        i += 1

    # Step 8: Combine employed bee and onlooker populations
    population = new_employed_bees + new_onlookers
```

```
    # Step 9: Abandon and search for new food sources
    for i in range(N):
      if(improvement[i]>p):
        abandoned_food_source = population[i]
        new_food_source = generate_new_food_source()
        population[i] = new_food_source

    # Step 10: Increment t
    t += 1

# Step 12: Output the optimal band combination
optimal_band_combination = population[0]

optimal_band_combination.sort(reverse=False)
print(optimal_band_combination)
print(len(optimal_band_combination))
```

```
from sklearn.svm import SVC
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import accuracy_score

X = data[:, optimal_band_combination]
y = data[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.8, random_state=42 , stratify=y)

# 2. Train an SVM Classifier
# Choose the SVM kernel and create the classifier
svm_classifier = SVC(kernel='linear')

# Train the classifier on the training data
svm_classifier.fit(X_train, y_train)

# 3. Test the SVM Classifier
# Make predictions on the testing data
y_pred = svm_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

class_accuracies = []
for class_label in np.unique(y_test):
    class_indices = np.where(y_test == class_label)
    class_accuracy = accuracy_score(y_test[class_indices],
y_pred[class_indices])
    class_accuracies.append(class_accuracy)
```

```
print(class_accuracies)
# Calculate average accuracy
average_accuracy = np.mean(class_accuracies)
print(f"Average Accuracy: {average_accuracy * 100:.2f}%")

kappa = cohen_kappa_score(y_test, y_pred)
print(f"Cohen's Kappa Coefficient: {kappa:.4f}")
```

```
from sklearn.ensemble import RandomForestClassifier
# Create a Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100,
random_state=42)  # You can adjust the hyperparameters

# Train the classifier on the training data
random_forest.fit(X_train, y_train)
# Make predictions on the testing data
y_pred = random_forest.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

class_accuracies = []
for class_label in np.unique(y_test):
    class_indices = np.where(y_test == class_label)
    class_accuracy = accuracy_score(y_test[class_indices],
y_pred[class_indices])
    class_accuracies.append(class_accuracy)

# Calculate average accuracy
average_accuracy = np.mean(class_accuracies)
print(f"Average Accuracy: {average_accuracy * 100:.2f}%")

print(class_accuracies)

kappa = cohen_kappa_score(y_test, y_pred)
print(f"Cohen's Kappa Coefficient: {kappa:.4f}")
```