Scalable Machine Learning

Neetha Ravva

I. Task-1

A. Chapter -1 Summary

Big Data revolves around handling extensive datasets, analyzing them to discover patterns or insights. Examples include data from systems like network intrusion detection, climate change monitoring, and public space surveillance. These datasets are typically unstructured, vast, complex, and difficult to manage and process. To simplify solutions, frameworks such as supervised learning and dimensionality reduction techniques like Principal Component Analysis are employed. Modern Big Data advancements leverage Hadoop for distributed file systems and machine learning, incorporating methods like decision tree learning and deep learning. Key Big Data concepts involve understanding data, along with physical, mathematical, and logical operations involved in knowledge processing.

Data is classified based on the insights or outcomes it yields, presented in formats like mathematical representations or tables, distinguishing between labeled and unlabeled data. Labeled data, where information is accessible, facilitates machine learning training. In contrast, unlabeled data, where information is concealed, serves for testing or validating machine learning models. The process of identifying patterns and deriving unknown statistical distributions creates a Knowledge/Response/Labeled Set. Critical operations like capture, storage, manipulation, and visualization play a vital role in enhancing or developing machine learning methods. Tools like Hadoop utilize MapReduce for addressing Big Data challenges in a distributed setup. Transforming data into knowledge necessitates mathematical and logical operations.

Three essential metrics to consider are the number of events (n), total events within a timeframe (t), and the total number of events or variables produced, known as the feature space (p). The feature space (p) aids in determining data dimensionality and managing complexity. The volume of data is denoted by the number of events over time, and the data rate (velocity) is the ratio of these events over time.

The machine learning paradigm aims to create optimal algorithms for datasets, requiring accurate data and domain mapping, known as Supervised Learning, which uses three datasets: training (labeled), validation, and testing. Modeling involves the mathematical and statistical structuring of data. Big Data algorithms include numerous built-in algorithms for understanding trained data and deriving potential outcomes or new algorithms, termed learning algorithms. In supervised learning, classes and their boundaries are predefined, facilitating learning through classification. Conversely, unsupervised learning deals with unknown classes, requiring the algorithm to identify classes and boundaries for analysis, known as clustering.

With labeled datasets, new analyzed data can be labeled by applying the rules from the training dataset. Classification is mathematically defined using vectors, which possess magnitude and direction, akin to velocity in moving from point A to B. Functions map datasets to outcomes. Techniques for classification include Support Vector Machine, Decision Tree, Random Forest, and Deep Learning. Clustering, which deals with unlabeled data, poses challenges in analysis and data formation but can be approximated using geometric patterns, among others, to label data. Clustering techniques include K-Means, Gaussian-Mixture, and Hierarchical Clustering, employing similar mathematical functions as classification.

B. Chapter -2 Summary

1) Big Data Essentials: Big data analytics tackles the ever-growing challenge of extracting meaningful insights from massive datasets. This information ocean is defined by three key dimensions: volume, velocity, and variety. Volume refers to the sheer size of the data, a constantly expanding sea that can overwhelm traditional analysis methods. Velocity describes the speed at which this data is generated and changes, demanding real-time processing capabilities. Finally, variety encompasses the diverse nature of the data, ranging from structured numbers to unstructured text and images.

To navigate this complex landscape, we rely on three crucial ”controllers”: Class, Feature, and Observation. Class represents the different categories or outcomes present in the data, but with increasing volume, identifying and classifying new classes becomes a significant challenge. This not only affects the processing power and storage needed but also contributes to potential performance slowdowns. Features, on the other hand, act as the independent variables influencing these outcomes. However, high-dimensional data with numerous features creates a complexity challenge, making it difficult to discern patterns and requiring techniques like dimensionality reduction. This, in turn, impacts processing needs and storage capacity. Finally, observations – the individual data points themselves – pose their own challenge. Organizing, processing, and analyzing massive numbers of observations can be overwhelming, impacting communication, data processing, and storage, especially in remote settings.

Understanding and effectively managing these controllers is essential for successful big data analysis. By mastering these concepts, we can unlock the true potential of this information ocean, transforming diverse data into valuable insights that drive informed decision-making.

2) Big Data Problems: The Class controller faces challenges due to the inherent unpredictability of Big Data. As the volume of Big Data expands, accurately classifying different classes becomes increasingly complex and uncertain. The

Feature controller encounters difficulties as the dimensionality of data escalates with growth, leading to issues with scalability. Similarly, the Observation controller struggles with the management, processing, and analysis of the data, which becomes more burdensome as the amount of data swells, pushing the limits of existing technological capabilities.

3) Big data Solutions: Solutions to these problems can be developed through specific techniques and technological advancements. In terms of techniques, employing efficient algorithms and models is key, such as using Supervised Machine Learning for class issues, feature hashing to address feature dimensionality, and stochastic gradient descent for observation-related problems. On the technology front, solutions can be categorized into systems and frameworks, with modern distributed file systems like Hadoop and contemporary programming frameworks such as the MapReduce Programming Model.

4) Big Data Classification: Classification in Big Data entails managing vast datasets, which includes gathering input data, comprehending the nature of the data, and organizing it effectively. Additionally, it involves setting up Big Data technology, which requires an understanding of the Big Data ecosystem tailored to specific hardware needs and the limitations of various controllers. Furthermore, it encompasses the creation of machine learning methods, focusing on grasping the intricacies of modeling and the algorithms employed.

5) Representational learning: Representation learning methods play a crucial role in analyzing and structuring data, particularly for the Feature Controller, by aiding in the reduction of data dimensionality. Statistical metrics such as the mean, standard deviation, and covariance are instrumental in numerically identifying patterns within the data. Visual tools like pie charts, histograms, and scatter plots offer insights into these patterns from a graphical perspective. Data manipulation techniques, including normalization and standardization, facilitate the extraction and comprehension of patterns. Representation learning adapts to the evolving characteristics of data, primarily focusing on data understanding rather than classification. The Cross-Domain Representation Learning Framework emerges as a valuable approach for conducting big data analytics.

6) Modeling Classifications: Describes a mapping of knowledge (response set) to data. takes into account the features of the class (imbalanced, incomplete, incorrect), the domain (dimensionality, sparsity, subspace), and the error characteristics (approximation, estimate, optimization).

7) Class Characteristics:

- **Unbalance:** A condition where one class has significantly fewer observations than another.
- **Inadequate:** Situations where certain class observations lack relevant information.
- **Inaccuracy:** Instances where the labels for class observations are incorrect.

8) Error Characteristics:

- **Error in approximation:** The discrepancy between actual models and theoretical assumptions.
- **Error in optimization:** The gap between optimal algorithms and those actually implemented.
- **Error in estimation:** Variability of model performance across different datasets.

9) Classification algorithms:

10) Training: Involves using labeled data to estimate, approximate, and optimize model parameters.

11) Validation: Utilizes cross-validation and early stopping to assess model efficacy with a different labeled dataset.

12) Testing: Employs another labeled dataset to evaluate the final model's performance metrics.

13) Big Data Scalability: Discusses challenges related to feature expansion and high-dimensional systems, and the importance of low-dimensional structures for scalability.

14) Extra Information: Highlights the importance of interdisciplinary collaboration, techniques for data selection, technological constraints, and provides a mathematical foundation for data to knowledge transformation.

C. Chapter-3 Summary

This chapter focuses on the key topics of analytics, the development of patterns, statistical metrics, graphical instruments, and machine learning strategies for categorizing large datasets. Section 3.1 delves into analytics, emphasizing the importance of comprehending statistical and geometric features to facilitate the emergence of patterns. Effective classification within big data necessitates precise segmentation of domains coupled with an understanding of statistical indices such as the mean, variance, covariance, and correlation. Graphical representations, including charts, histograms, and scatter plots, are crucial for visually identifying patterns. Familiarity with statistical methods like standardization, normalization, linear transformation, and orthogonalization plays a vital role in the progression and identification of patterns.

1) Research: Research is centered around employing statistical and geometric metrics to create machine learning models tailored for big data categorization. Central inquiries involve examining the traits of datasets, the distinctness of classes, issues of data imbalance, challenges related to scalability, the aspect of dimensionality, and the identification of low-dimensional configurations. In the context of big data, it is crucial to define classes for clusters of observations, particularly within batch learning settings.

2) Data handling: The datasets utilized for simulations range from straightforward to intricate examples. These datasets undergo randomization processes to ensure they are balanced, precise, and comprehensive. Fundamental statistical tools and visual aids such as the mean, median, mode, variance, along with pie charts and histograms, are employed to identify and extract patterns within these datasets.

3) Analytics Foundations: Recognizing Trends: Fundamentals of analytics, including geometrical and statistical features, are essential to understanding how patterns emerge in large data sets. Effective domain division strategies for big data classification involve the use of visual aids like graphs, histograms, and scatter plots in addition to statistical metrics like counting, mean, variance, covariance, and correlation.

4) Data Set Selection: Significance for Categorization: Big data classification can only be understood in part by careful data set handling and selection. Initially, two straightforward data sets—Carpet Floor and Hardwood Floor—are selected because they meet Gaussian models with little inseparability problems. Through randomization, imbalances, errors, and incompleteness are addressed in the process.

5) Statistical Analysis: Statistical methods and pattern recognition tools significantly improve the accuracy of machine learning classifications. Metrics such as frequency counts, averages, and variances play a crucial role in data characterization, distinguishing between class variances, and tackling issues like data imbalance. These statistical measures are instrumental in analyzing features and refining classification methodologies. The statistical domain employs graphical representations like histograms, scatter diagrams, and measures of skewness for enhanced visual analytics. Histograms offer insights into the distribution of data and support hypothesis testing, while skewness helps in classifying and uncovering data attributes through its asymmetry. Scatter diagrams are useful for illustrating data's central tendencies, variability, and relationships, facilitating the reduction of dimensionality and the uncovering of underlying structures. This emphasizes the significance of statistical metrics, geometric considerations, and visual aids in the comprehension, analysis, and categorization of complex data patterns, alongside addressing the challenges and methodologies associated with data handling, feature examination, and classification optimization within vast datasets.Grasping the characteristics of data is significantly enhanced by the application of statistical and geometric methods for pattern detection, including the mean, variance, covariance, and correlation. The identification of patterns is facilitated through visual tools like histograms and scatter plots. Employing statistical techniques to derive valuable insights from images is illustrated through examples from practical scenarios. The nature of big data patterns is dynamic, posing challenges in developing accurate supervised learning models. Examples such as MyPrismaColors and Biltmore Estate underscore the intricacies involved in recognizing patterns. The split-merge-split approach incorporates methods for data augmentation to methodically study the progression of these patterns.The split-merge-split technique is employed to emulate big data analysis, utilizing a model for expanding data. This approach features four principal applications: orthogonalization, linear transformation, standardization, and normalization. For the purpose of observing the evolution of patterns, the methodology introduces models for expanding data that are parameterized, including techniques like mean-shift, Gaussian weights, and Gaussian rise.Class characteristics, including incomplete, inaccurate, and unevenly distributed data, significantly affect pattern deformation. The influence of these factors on data patterns is demonstrated by employing the Hardwood Floor and Carpet Floor datasets as examples.Classification inaccuracies, such as errors in approximation, estimation, and optimization, underscore challenges in modeling. These errors emphasize the importance of accurate algorithm selection and precise parameter estimation for achieving effective classification outcomes.The focus currently lies in revealing critical low-dimensional patterns hidden within the data. Demonstrations of reducing data from three dimensions to two illustrate the importance of data simplification in comprehending and visualizing underlying patterns.

II. Task -2

I have used the Google collab environment. Google collab is the cloud hosted environment so it does not need any commands.

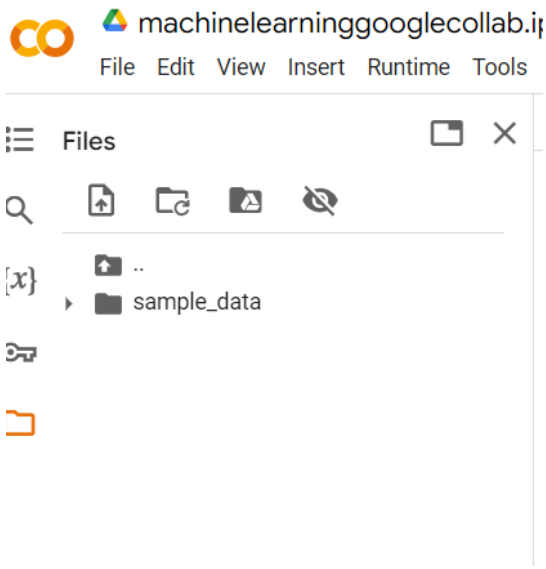

Fig. 1: Google collab environment

III. Task -3

A. Input Images

These are the images we are going to use in this assignment.

Fig. 2: Cardinal.

Fig. 3: Sparrow

Fig. 4: Mangroove cuckoo

IV. Task-4 Channel display

In the task 4 need to convert the original-image into rgb channels and to gray scale and we will be displaying their dimensions.

A. Python Code and its Explanation

1) **Import Libraries:**

```
import cv2
from matplotlib import pyplot as plt
import os
```

2) **Code to Display RGB Channels Function**

```
def read_and_display_Images(images):
    for im in images:
        read_image = cv2.imread(im,cv2.
            IMREAD_COLOR)
        rgb_color = cv2.cvtColor(
            read_image,cv2.COLOR_BGR2RGB)
        filename = os.path.basename(im)
        print(f'Original␣image:{filename}'
            )
        # Display the original image
        plt.figure(figsize=(12, 4))
        plt.subplot(141)
        plt.imshow(rgb_color)
        plt.title('Original␣Image')

        r, g, b = cv2.split(rgb_color)

        plt.subplot(142)
        plt.imshow(r, cmap='Reds')
        plt.title('Red␣Channel')

        plt.subplot(143)
        plt.imshow(g, cmap='Greens')
        plt.title('Green␣Channel')

        plt.subplot(144)
        plt.imshow(b, cmap='Blues')
        plt.title('Blue␣Channel')

        plt.show()
```

3) **Code to display Grayscale**

```
def display_color_grayscale_and_dimensions
    (images):
    for image in images:
        read_image = cv2.imread(image,cv2.
            IMREAD_COLOR)
        rgb_color = cv2.cvtColor(
            read_image,cv2.COLOR_BGR2RGB)
        # Converting the color image to
            grayscale
        gray_scale_image = cv2.cvtColor(
            rgb_color, cv2.COLOR_BGR2GRAY)
        # Display the color and grayscale
            images
        plt.figure(figsize=(10, 5))
        plt.subplot(121)
        plt.imshow(rgb_color)
        plt.title('Color␣Image')
        plt.axis('off')

        plt.subplot(122)
        plt.imshow(gray_scale_image, cmap=
            'gray')
        plt.title('Grayscale␣Image')
        plt.axis('off')

        plt.tight_layout()
        plt.show()

        # Printing the dimensions of the
            image1
        print(f"Color␣image␣dimensions␣(
            Height␣x␣Width):␣␣{rgb_color.
```

```
            shape[0]}␣x␣{rgb_color.shape
            [1]}")
24      print(f"Grayscale␣image␣dimensions
            :␣(Height␣x␣Width):␣␣{
            gray_scale_image.shape[0]}␣x␣{
            gray_scale_image.shape[1]}")
```

4) **Main Section**

```
1  images =[r'/content/Machine␣learning␣
       assignment/Image1.jpg', r"/content/
       Machine␣learning␣assignment/Image2.jpg
       ",r"/content/Machine␣learning␣
       assignment/Image3.jpg"]
2  read_and_display_Images(images)
3  display_color_grayscale_and_dimensions(
       images)
```

Explanation: The above code uses the OpenCV library to read, process, and display images. It also uses Matplotlib for visualization. Here's a brief explanation of each function:

- `read_and_display_Images(images)`: This function reads images from the provided list of file paths (`images`). For each image, it reads the image using OpenCV's `cv2.imread()` function, converts the color space from BGR to RGB using `cv2.cvtColor()`, and then displays the original image along with its individual RGB channels using Matplotlib's `imshow()` function. The `plt.subplot()` function is used to arrange the plots in a single figure.
- `display_images_grayscale_and_dimensions(images)`: This function also reads images from the provided list of file paths (`images`). For each image, it reads the image using OpenCV's `cv2.imread()` function, converts the color space from BGR to RGB using `cv2.cvtColor()`, and then converts the RGB image to grayscale using `cv2.cvtColor()`. It displays the color image and its grayscale counterpart side by side using Matplotlib. Additionally, it prints the dimensions (height and width) of both the color and grayscale images.

B. OUTPUT

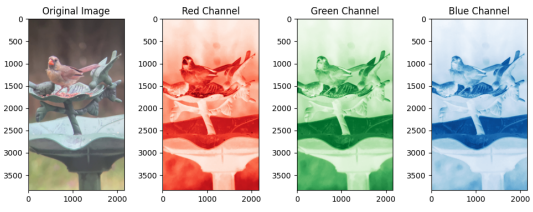

Fig. 5: Gray Scale Image of cardinal

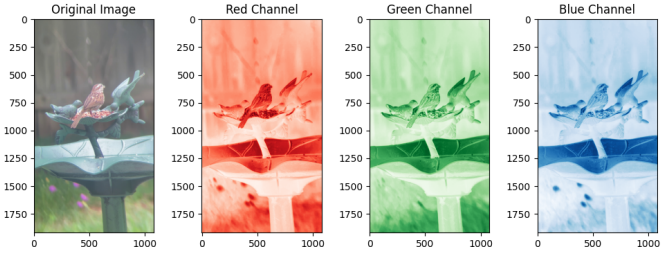

Fig. 6: RGB Image of Sparrow

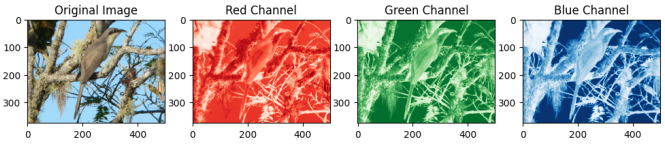

Fig. 7: RGB Image of Mangroove cuckoo

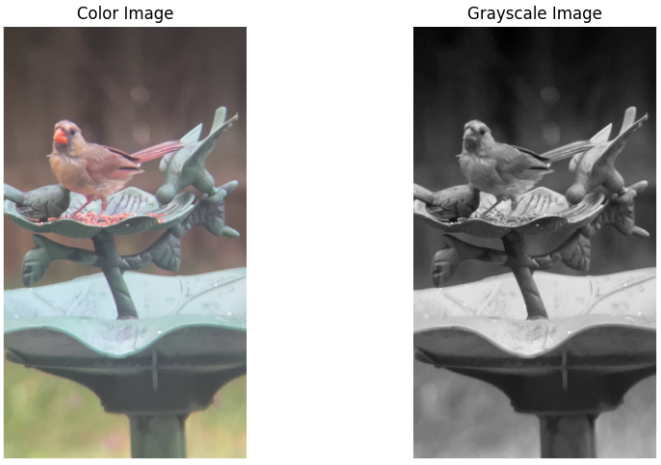

Fig. 8: Gray-scale and color image of cardinal with dimensions (Height x Width): 3840 x 2160

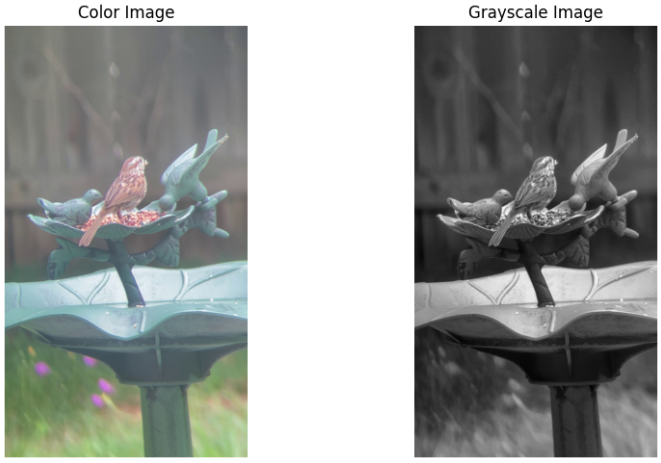

Fig. 9: Grayscale and color image of sparrow with dimensions (Height x Width): 1920 x 1080

Fig. 10: Gray-scale and color image of mangrove cuckoo with dimensions(Height x Width): 375 x 500

V. Task-5 Resizing

Task 5 section is about resizing grayscale images to a height of 256 pixels while approximately maintaining the aspect ratio and ensuring the width is divisible by 16.

A. Python Code and its Explanation

```
    \item \
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Define a function to resize the grayscale image
def resizing_image(image, desired_height):
    # Get the original height and width
    original_height, original_width = image.shape[:2]

    # Calculate the new width while maintaining the aspect ratio
    aspect_ratio = original_width / original_height
    desired_width = int(desired_height * aspect_ratio)

    # Ensure the width is divisible by 16
    if desired_width % 16 != 0:
        desired_width = ((desired_width + 15) // 16) * 16

    # Resize the image using the calculated dimensions
    resized_image = cv2.resize(image, (desired_width, desired_height))

    return resized_image

# Example usage:
# Load a grayscale image (replace with your image path)
image_paths = [r'/content/Machine␣learning␣assignment/Image1.jpg', r"/content/Machine␣learning␣assignment/Image2.jpg",r"/content/Machine␣learning␣assignment/Image3.jpg"]
for image_path in image_paths:
    grayscale_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Define the desired height (256 pixels)
    desired_height = 256

    # Resize the image
    resized_image = resizing_image(grayscale_image, desired_height)

    # Display the resized image and its dimensions
    plt.imshow(resized_image, cmap='gray')
    plt.title(f"Resized␣Image␣Dimensions␣(Height␣x␣Width):␣{resized_image.shape[0]}␣x␣{resized_image.shape[1]}")
    plt.axis('off')
    plt.show()
```

Explanation:

1) Firstly the code imports the required libraries like `cv2` for image processing, `numpy` for numerical operations, and `matplotlib.pyplot` for visualization.
2) We need to define resizing_image function and it-takes two arguments: `image` (a grayscale image) and `desired_height` (the target height for resizing). It calculates the aspect ratio of the original image and uses it to compute the corresponding width to maintain the aspect ratio. It ensures that the calculated width is divisible by 16 . It resizes the input image to the specified dimensions (`desired_height` x calculated width) using OpenCV's `cv2.resize` function. The resized image is returned as the output.
3) A list of image file paths (imagepaths) is defined, containing the paths to the images you want to resize (image1.jpg, image2.jpg, image3.jpg). A loop iterates over the image paths, and for each image, it performs the following steps: Reads the image as grayscale using cv2.imread. Specifies a desired height for resizing (in this case, 256 pixels). Calls the resizeing_image function to resize the grayscale image to the desired height. Displays the resized image using cv2.imshow.

B. OUTPUT

Fig. 11: Resized Gray-scale Image of cardinal

Fig. 12: Resized Gray-scale Image of sparrow

Fig. 13: Resized Gray-scale Image of a Mangrove cuckoo

VI. TASK-6 EXTRACTING BLOCK FEATURE VECTORS

The main aim of this task is to Split each image into non-overlapping 16x16 pixel blocks. Convert 16X16 blocks into 256-dimensional vectors. Labeling each feature vector with 0, 1, or 2, corresponding to the first, second, and third images, respectively. Compile these vectors into a DataFrame, where each row represents a feature vector from the images. Save this DataFrame to a CSV file.

A. Python Code Explanation

1) **Generate Block-Feature Vectors**

```
import cv2
import numpy as np
import pandas as pd


# Function to generate block-feature vectors from an image
def generate_block_feature_vectors(image, block_size=16):
    feature_vectors = []
    height, width = image.shape

    for y in range(0, height, block_size):
        for x in range(0, width, block_size):
            block = image[y:y+block_size, x:x+block_size]
            if block.shape == (block_size, block_size):
                # Convert the block to a 1D vector (256 elements)
                block_vector = block.flatten()
                feature_vectors.append(block_vector)

    return feature_vectors

# Load your three grayscale images (replace with your image paths)
```

```
image_paths = [r'/content/Machine␣learning␣assignment/Image1.jpg',r"/content/Machine␣learning␣assignment/Image2.jpg",r"/content/Machine␣learning␣assignment/Image3.jpg"]
images = []

for image_path in image_paths:
    grayscale_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    # Define the desired height (256 pixels)
    desired_height = 256
    # Resize the image
    resized_image = resizing_image(grayscale_image, desired_height)
    if resized_image is not None:
        images.append(resized_image)
    else:
        print(f"Warning:␣Unable␣to␣access␣the␣image␣path{image_path}")
#for path in image_paths:
    #image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    #if image is not None:
       # images.append(image)
    #else:
       # print(f"Warning: Unable to read image at {path}")

# Assign labels to images (0, 1, 2 for the first, second, and third images)
labels = [0, 1, 2]

# Generate block-feature vectors for each image
block_size = 16
entire_feature_vectors = []
entire_labels = []

for i, image in enumerate(images):
    feature_vectors = generate_block_feature_vectors(image, block_size)
    entire_feature_vectors.extend(feature_vectors)
    entire_labels.extend([labels[i]] * len(feature_vectors))

# Create a Pandas DataFrame to store the feature vectors
df = pd.DataFrame(entire_feature_vectors)

# Add a column for labels
df['Label'] = entire_labels

# Export the DataFrame to a CSV file
df.to_csv('/content/Machine␣learning␣assignment/block_feature_vectors1.csv', index=False)

# Display the DataFrame
print(df)
```

Explanation:

1) Firstly, we are importing necessary libraries: 'cv2' (OpenCV for image processing), 'numpy' (for numerical operations), and 'pandas' (for data handling).
2) generate_block_feature_vectors function accepts an image as input and generates block feature vectors from it. It divides the image into non-overlapping blocks of a specified size (default is 16x16 pixels). For each block, it flattens the pixel values into a 1D vector of 256 elements. The function returns a list containing these block feature vectors.
3) The code loads three grayscale images from the provided file paths. Grayscale images are preferred for simplicity as they consist of a single channel.
4) Each image is assigned a label (0, 1, and 2) to identify them.
5) The code iterates through the loaded images. For each image, it invokes the 'generate_block_feature_vectors' function to extract block feature vectors. These vectors are stored in the 'entire_feature_vectors' list, while their corresponding labels are stored in the 'entire_labels' list. The labels are repeated for each block within an image.
6) A Pandas DataFrame ('df') is created to store the block feature vectors. Each row in the DataFrame represents a block feature vector, and there is an additional column named 'Label' to store the corresponding label.
7) The DataFrame is exported to a CSV file named 'block_feature_vectors.csv' without including the row indices (index=False).
8) Finally, the DataFrame is printed, showing the block feature vectors along with their corresponding labels.stored in a Pandas DataFrame along with corresponding labels.

VII. Task-7 Generating sliding block feature vectors

The primary aim of this task is to process a series of images by segmenting each one into overlapping blocks of 16x16 pixels and subsequently converting these blocks into feature vectors. For each of the first three images processed, the corresponding feature vectors are assigned labels 0, 1, and 2, respectively. These feature vectors, along with their assigned labels, are then organized into a DataFrame, with each vector occupying a row. The final step involves saving this organized data into a CSV file.

A. Python Code Explanation

1) **Generate Sliding Block-Feature Vectors.:**

```
import cv2
import numpy as np
import pandas as pd

# Function to generate sliding block-feature vectors from an image
def generate_sliding_block_feature_vectors(image, block_size=16):
    feature_vectors = []
    height, width = image.shape

    for y in range(0, height - block_size + 1, 1):  # I am taking the slider as 1.
        for x in range(0, width - block_size + 1, 1):
```

```
            block = image[y:y+block_size,
                x:x+block_size]
            if block.shape == (block_size,
                block_size):
                # Convert the block to a 1
                    D vector (256 elements
                    )
                block_vector = block.
                    flatten()
                feature_vectors.append(
                    block_vector)

    return feature_vectors

# Load your three grayscale images (
    replace with your image paths)
image_paths = [r'/content/Machine␣learning
    ␣assignment/Image1.jpg', r"/content/
    Machine␣learning␣assignment/Image2.jpg
    ",r"/content/Machine␣learning␣
    assignment/Image3.jpg"]
images = []

for image_path in image_paths:
    grayscale_image = cv2.imread(
        image_path, cv2.IMREAD_GRAYSCALE)
    # Define the desired height (256
        pixels)
    desired_height = 256
    # Resize the image
    resized_image = resizing_image(
        grayscale_image, desired_height)
    if resized_image is not None:
        images.append(resized_image)
    else:
        print(f"Warning:␣Unable␣to␣access␣
            the␣image␣path{image_path}")


# Assign labels to images (0, 1, 2 for the
     first, second, and third images)
labels = [0, 1, 2]

# Generate sliding block-feature vectors
    for each image
block_size = 16
entire_feature_vectors = []
entire_labels = []

for i, image in enumerate(images):
    feature_vectors =
        generate_sliding_block_feature_vectors
        (image, block_size)
    entire_feature_vectors.extend(
        feature_vectors)
    entire_labels.extend([labels[i]] * len
        (feature_vectors))

# Create a Pandas DataFrame to store the
    feature vectors
df = pd.DataFrame(entire_feature_vectors)

# Add a column for labels
df['Label'] = entire_labels

# Export the DataFrame to a CSV file
df.to_csv('/content/Machine␣learning␣
    assignment/
    sliding_block_feature_vectors.csv',
    index=False)

# Display the DataFrame
print(df)
```

Explanation:

1) Firstly it imports the required libraries, including OpenCV ('cv2'), NumPy ('np'), and Pandas ('pd').
2) A function named generate_sliding_block_feature_vectors(image, blocksize) is defined. This function takes an input image and generates sliding block feature vectors from it. It slides a window of size blocksize (default 16) across the image in both horizontal and vertical directions with the slider of 1. For each position of the window, it extracts a block of pixels of size blocksize x blocksize. If the extracted block is of the correct size, it flattens the block into a 1D vector (256 elements) and appends it to a list of feature vectors. The function returns this list of feature vectors.
3) Grayscale images specified by their file paths (`imagepaths`) were loaded using OpenCV's `cv2.imread` function. The loaded images are appended to a list called `images`.
4) Labels are assigned to the images. Here I am using '[0, 1, 2]' to represent the first, second, and third images, respectively.
5) Now it iterates over each loaded image and calls the `generateslidingblockfeaturevectors` function to generate sliding block feature vectors. The extracted feature vectors are added to a list called `allfeaturevectors`, and their corresponding labels (image indices) are added to a list called `alllabels`.
6) A Pandas DataFrame (`df`) is constructed to store the feature vectors. Each row of the DataFrame represents a feature vector, and there are columns for each element in the feature vector. Additionally, a column labeled 'Label' is added to store the corresponding image labels.
7) The Pandas DataFrame (`df`) is exported to a CSV file named `slidingblockfeaturevectors.csv` using the `to_csv` method. The `index=False` argument prevents saving the DataFrame index as a column in the CSV file.
8) (`sliding_block_feature_vectors.csv`) containing sliding block feature vectors extracted from the input grayscale images, with each vector associated with a label indicating which image it originated from.

VIII. TASK-8 Statistical descriptors and visualization

In task 8 we will be deriving the statistical components and histogram is used

A. Python code and its explanation

1) **statistical descriptors**

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd

# Load the feature space CSV file
block_vectors_df = pd.read_csv('/content/Machine learning assignment/block_feature_vectors1.csv')
statistics_info = block_vectors_df.describe()
print(statistics_info)

# Display basic statistical information
print("Number of observations:", len(df))
print("Dimension of the data:", df.shape)
print("Mean of each feature:")
print(df.mean())

# Calculate statistical descriptors
mean_features = np.mean(block_vectors_df, axis=0)
std_features = np.std(block_vectors_df, axis=0)
skewness = np.mean(((block_vectors_df - mean_features) / std_features) ** 3, axis=0)
kurtosis = np.mean(((block_vectors_df - mean_features) / std_features) ** 4, axis=0)
# Select features for histogram

selected_features = [0, 1, 2]


class_counts = block_vectors_df['Label'].value_counts()
print(f"Class Distribution:\n{class_counts}")

# Check for missing values
missing_values = block_vectors_df.isnull().sum()
print("Missing values:")
print(missing_values)

# Check dataset size
dataset_size = len(block_vectors_df)
print(f"Dataset Size: {dataset_size}")

# Check feature dimensionality
feature_dimensionality = block_vectors_df.shape[1] - 1 # Exclude the 'Label' column
print(f"Feature Dimensionality: {feature_dimensionality}")

# Histograms for selected features
block_vectors_df.iloc[:, selected_features].hist(bins=20, figsize=(12, 8), color='blue', alpha=0.7)
plt.suptitle('Histograms plot', fontsize=16)
plt.show()
```

Explanation:

1) Necessary libraries were imported and the script uses `matplotlib.pyplot`, `seaborn` for plotting, `pandas` for data handling, and assumes the use of `numpy` for statistical calculations.
2) A CSV file is read into the `block_vectors_df` DataFrame.
3) Basic statistics such as mean, standard deviation, skewness, and kurtosis are computed to understand data distribution.
4) Class Distribution Check is performed to evaluate the balance across different classes by counting instances per class.
5) Missing values were tested because they are crucial for data preprocessing.
6) Histograms were generates for selected features to visualize the distribution and identify any patterns or anomalies.

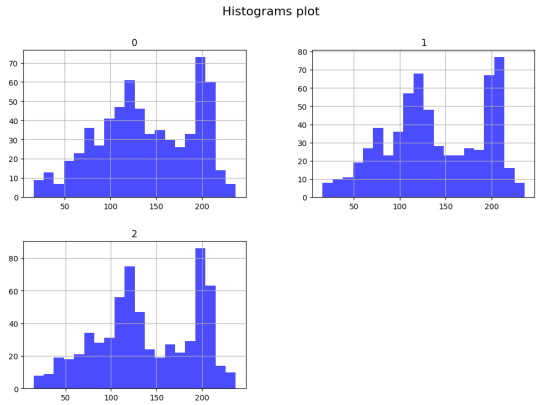

Fig. 14: Histogram of the selected features

1) Statistical Analysis::

- **Is the dataset imbalanced, inaccurate, or incomplete?**
  - The dataset appears to be imbalanced as there are significantly more samples belonging to class 2 compared to classes 0 and 1.
  - There are no missing values, indicating that the dataset is complete.
  - However, without further context, it's difficult to assess accuracy.
- **Is it a trivial data or possibly a big data? Does it have a scalability problem?**
  - With 640 observations, the dataset is not trivial but also not very large, indicating it's not a big data problem.
  - The dataset size is manageable, suggesting that scalability may not be a significant concern.
- **Is it high dimensional?**
  - The feature space has a dimensionality of 256, which is relatively high.
- **Do you need to standardize? Do you need to normalize? How do they affect the data characteristics?**
  - It may be necessary to standardize or normalize the features, especially given their varying means and standard deviations across different dimensions.
  - Standardization and normalization could help improve the performance of certain algorithms and ensure fair comparison between features.

- Standardization scales the features such that they have a mean of 0 and a standard deviation of 1, which can be beneficial for algorithms that rely on distance metrics or optimization techniques.
- Normalization scales the features to a range between 0 and 1, which can be useful for algorithms that are sensitive to the scale of features or when dealing with features with different units.

IX. TASK-9 Constructing feature space

In this task, block-feature vectors are produced and labeled DataFrames are established for each image. Subsequently, data frames were merged and the feature vectors are randomized, and the resulting datasets, along with their shuffled variants, are exported to CSV files.

A. Python Code

1) **Import Libraries:**

```
import pandas as pd
import cv2
image_paths = [r'/content/Machine␣learning␣assignment/Image1.jpg',r"/content/Machine␣learning␣assignment/Image2.jpg",r"/content/Machine␣learning␣assignment/Image3.jpg"]


grayscale_image_1 = cv2.imread(image_paths[0], cv2.IMREAD_GRAYSCALE)
image_1_gray_resized = resizing_image(grayscale_image_1, desired_height)
#print(image_1_gray_resized.shape)


block_features_0_df = pd.DataFrame(generate_block_feature_vectors(image_1_gray_resized, block_size=16))
block_features_0_df['label'] = 0
print(block_features_0_df)
block_features_0_df.to_csv('/content/Machine␣learning␣assignment/image0.csv')

grayscale_image_2 = cv2.imread(image_paths[1], cv2.IMREAD_GRAYSCALE)
image_2_gray_resized = resizing_image(grayscale_image_2, desired_height)
print(image_2_gray_resized.shape)

block_features_1_df = pd.DataFrame(generate_block_feature_vectors(image_2_gray_resized,block_size=16))
block_features_1_df['label'] = 1
print(block_features_1_df)
block_features_1_df.to_csv('/content/Machine␣learning␣assignment/image1.csv')

grayscale_image_3 = cv2.imread(image_paths[2], cv2.IMREAD_GRAYSCALE)
image_3_gray_resized = resizing_image(grayscale_image_3, desired_height)
print(image_3_gray_resized.shape)

block_features_2_df = pd.DataFrame(generate_block_feature_vectors(image_3_gray_resized,block_size=16))
block_features_2_df['label'] =2
print(block_features_2_df)
block_features_1_df.to_csv('/content/Machine␣learning␣assignment/image2.csv')
```

Explanation: Grayscale image has been resized , block feature vectors were generated for each image and they were stored in csv files like image0.csv, image1.csv and image2.csv

2) **Merge and randomize function**

```
# Function to merge and randomize feature vectors
def merge_and_randomize(dataframes):
    merged_data = pd.concat(dataframes, ignore_index=True)
    random_indices = np.random.permutation(len(merged_data))
    merged_data = merged_data.iloc[random_indices]
    return merged_data
```

Explanation: The function `merge_and_randomize` merges the data frames of the block vectors and generates the randomized data.

```
inputfiles = [block_features_0_df, block_features_1_df]
block_0_1_df = merge_and_randomize(inputfiles)
print(block_0_1_df)
block_0_1_df.to_csv('/content/Machine␣learning␣assignment/image01.csv',index=False)
inputfiles =[block_features_0_df, block_features_1_df, block_features_2_df]
block_0_1_2_df = merge_and_randomize(inputfiles)
print(block_0_1_2_df)
block_0_1_2_df.to_csv('/content/Machine␣learning␣assignment/image012.csv', index=False)
```

Explanation: The data frames of the block vector features were merged, randomized and stored to csv files(image01.csv and image012.csv).

X. TASK-10 Displaying the sub spaces

In this task we display the feature spaces

A. Python Code and it's Explanation

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Import for 3D plotting
import numpy as np

# Load data from CSV files
# Replace 'image0.csv', 'image1.csv', 'image2.csv' with your actual file paths
```

```
df0 = pd.read_csv('/content/Machine␣learning␣assignment/image0.csv')
df1 = pd.read_csv('/content/Machine␣learning␣assignment/image1.csv')
df2 = pd.read_csv('/content/Machine␣learning␣assignment/image2.csv')




# Assuming the CSV files have columns named 'feature1', 'feature2', 'feature3', and 'bird_label'
# Replace these column names with your actual feature and label column names

# Step 2: Select Two Features
features_selected_2d = [0,1]

# Step 3: Create a 2D Plot
plt.figure(figsize=(8, 6))

# Plot data from data0
plt.scatter(df0.iloc[:, features_selected_2d[0]], df0.iloc[:, features_selected_2d[1]], label='Bird␣0', marker='o')

# Plot data from data1
plt.scatter(df1.iloc[:, features_selected_2d[0]], df1.iloc[:, features_selected_2d[1]], label='Bird␣1', marker='x')

# Label axes and add a legend
plt.xlabel(features_selected_2d[0])
plt.ylabel(features_selected_2d[1])
plt.title('Two-Dimensional␣Feature␣Space')
plt.legend()
plt.grid(True)

# Show the 2D plot
plt.show()

# Step 4: Select Three Features
features_selected_3d = [0,1,2]

# Step 5: Create a 3D Plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# Plot data from data0
ax.scatter(df0.iloc[:,features_selected_3d[0]], df0.iloc[:,features_selected_3d[1]], df0.iloc[:,features_selected_3d[2]], label='Bird␣0', marker='o')

# Plot data from data1
ax.scatter(df1.iloc[:,features_selected_3d[0]], df1.iloc[:,features_selected_3d[1]], df1.iloc[:,features_selected_3d[2]], label='Bird␣1', marker='x')

# Plot data from data2
ax.scatter(df2.iloc[:,features_selected_3d[0]], df2.iloc[:,features_selected_3d[1]], df2.iloc[:,features_selected_3d[2]], label='Bird␣2', marker='^')

# Label axes and add a legend
ax.set_xlabel(features_selected_3d[0])
ax.set_ylabel(features_selected_3d[1])
ax.set_zlabel(features_selected_3d[2])
ax.set_title('Three-Dimensional␣Feature␣Space')
ax.legend()
ax.grid(True)

# Show the 3D plot
plt.show()
```

*Explanation:*Necessary libraries were imported.*Pandas* for data manipulation. *Matplotlib* for plotting. *mpltoolkits.mplot3d* for 3D visualization.

a) Feature vectors for various birds are loaded from three CSV files into separate DataFrames:
   - `image0.csv` into `df0`.
   - `image1.csv` into `df1`.
   - `image2.csv` into `df2`.

b) Selection of features for plotting is based on column indices:
   - For 2D plots, features at indices 0 and 1 are selected.
   - For 3D plots, features at indices 0, 1, and 2 are chosen.

c) A 2D plot is generated to visualize selected features from `df0` and `df1`, differentiating bird classes with unique markers and including axis labels and a legend.

d) A 3D plot is similarly constructed to display selected features from `df0`, `df1`, and `df2`, with each bird class represented by distinct markers. Axis labels and a legend are added for clarity.

e) This visualization strategy effectively facilitates the comparison and analysis of bird species based on their feature vectors, demonstrating class separability in multidimensional spaces.

B. OUTPUTS

XI. TASK-11 HANDLING MULTIPLE IMAGES

In this task, we extract block feature vectors from photographs from 'Birds' folder using OpenCV and scikit-learn in Google Colab. We then resize the images to 16x16 pixels and save the resulting feature space to a CSV file.

A. Python Code Explanation

```
import zipfile
import os

# Path to the zip file
zip_file_path = "/content/Machine␣learning␣assignment/Birds.zip"

# Directory to extract the contents
extract_to_directory = "/content/data"
```

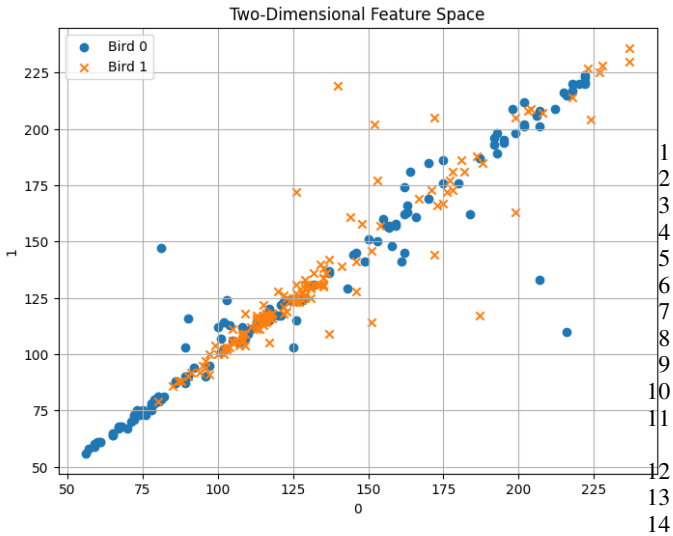

Fig. 15: 2D Feature Space of the Selected Features

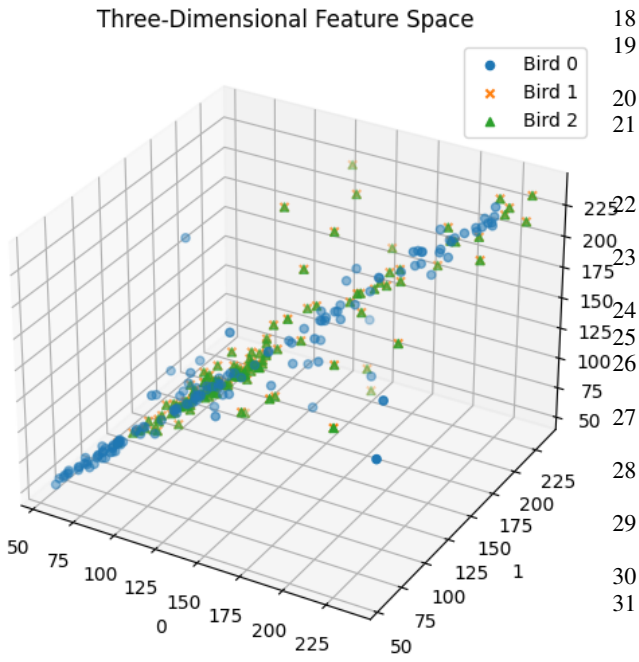

Fig. 16: 3D Feature Space of the Selected Features

```python
# Check if the directory already exists, if not, create it
if not os.path.exists(extract_to_directory):
    os.makedirs(extract_to_directory)

# Unzip the file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_directory)

# List the contents of the extracted directory
extracted_files = os.listdir(extract_to_directory)
print("Files extracted successfully:")
print(extracted_files)
```

Explanation: This part of the code imports the necessary libraries, including zipfile, os. Then the code unzips the zipped bird folder and stores it to the specified location.

Function to Extract Features block vectors from the directory.

```python
import cv2
import os
import numpy as np
import pandas as pd



# Directory containing the images
image_folder = '/content/data/Birds'

# List to store feature vectors from all images
all_feature_vectors = []
all_labels = []

# Block size for feature extraction
block_size = 16

# Iterate through all images in the folder
for root, dirs, files in os.walk(image_folder):
    for file in files:
        if file.endswith(('.jpg', '.png', '.jpeg')):  # Filter by image file extensions
            image_path = os.path.join(root, file)
            image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

            if image is not None:
                # Generate block-feature vectors for each image
                feature_vectors = generate_block_feature_vectors(image, block_size)
                all_feature_vectors.extend(feature_vectors)

                # Assign a label based on the image file name or any other criteria
                label = 0  # Replace with logic to assign labels
                all_labels.extend([label] * len(feature_vectors))

# Create a DataFrame for the feature vectors and labels
feature_space = pd.DataFrame(all_feature_vectors)
feature_space['Label'] = all_labels

# Save the feature space to a CSV file
feature_space.to_csv('feature_space.csv', index=False)

# Display the first few rows of the feature space (optional)
print("Feature Space:")
print(feature_space.head())
```

Explanation:

1) Identifies the directory (`imagefolder`) containing the images and sets the block size (`blocksize`) for feature extraction.
2) Iterates through each file in `imagefolder`, loading images with accepted extensions (.jpg, .png, .jpeg) in grayscale mode using OpenCV (`cv2.imread`).
3) Invokes the `generate_block_feature_vectors` function to extract feature vectors from each valid image based on the specified block size, adding these vectors to the `all_feature_vectors` list.
4) Assigns an initial label (`label = 0`) to each feature vector, which can be updated with tailored labeling logic based on image features or filenames.
5) Gathers all feature vectors into a Pandas DataFrame named `feature_space`, incorporating a 'Label' column for the assigned labels.
6) Exports the `feature_space` DataFrame to a CSV file (`feature_space.csv`) using the `to_csv` method, preparing it for further analysis, machine learning, or visualization.

XII. Task-12 Effects of block size

A. Choosing the Appropriate Block Size: Influence on Vector Quantity and Feature Space Complexity

The decision on block size plays a pivotal role in defining the complexity of the feature space and the number of vectors it contains during the transformation of raw data into a structured feature space. This section delves into the nuances of block size selection and its implications for the performance of classifiers operating within this transformed domain.

B. Dimensionality of the Feature Space

The size of the blocks chosen for data processing significantly impacts the dimensionality of the resulting feature space. Using larger block sizes can yield more complex representations that capture finer details and nuances within the data. However, this increase in complexity is accompanied by a rise in dimensionality, which can pose challenges in terms of computational efficiency and potentially exacerbate the curse of dimensionality. Conversely, opting for smaller block sizes may simplify the feature space but risk overlooking certain subtle aspects of the data.

Choosing the optimal block size requires a thoughtful equilibrium between computational feasibility and the level of detail. It's essential to consider the specific characteristics of the data and the requirements of downstream tasks, such as regression or classification, to make this decision.

C. Quantity of vectors

The quantity of vectors in the transformed domain is influenced by the size of the blocks used. Larger blocks tend to generate a smaller number of vectors, each encompassing a broader segment of the original dataset. On the other hand, smaller blocks lead to a higher density of vectors, potentially offering a more detailed capture of local variations within the data.

The classifier's capacity to discern and analyze data intricacies goes beyond merely the count of vectors. In scenarios with fewer vectors, derived from larger blocks, classifiers might struggle to detect subtle patterns. Conversely, a high number of vectors, resulting from smaller blocks, could introduce noise or redundant information, complicating the classification process.

D. Effect on the Classifier

Choosing the right block size significantly influences the performance of the classifier, responsible for analyzing the domain through the altered data. A classifier trained on a feature space with high dimensionality might exhibit heightened sensitivity to variations, increasing the risk of overfitting. Conversely, a classifier working within a lower-dimensional space could demonstrate greater robustness but potentially miss critical details.

The influence of block size on the classifier extends beyond mere model complexity, computational demands, and the capacity for generalization. Optimal block size selection, aligned with the specific objectives of the classification task and the data characteristics, is crucial for balancing efficiency and precision.

In conclusion, the process of selecting the appropriate block size involves a thorough evaluation of its impact on the feature space's dimensionality, the resulting vector count, and the consequent effects on classifier performance. Striking the right balance is key to effectively uncovering relevant patterns within the data while navigating the challenges associated with high dimensionality and vector density.

XIII. Assignment 2

A. Task 1 splitting the data

I generated training and testing data sets for sliding block vectors and block vector files for both 2 class and 3 class classifiers. Below are the steps which I performed.

I imported necessary libraries for splitting the data:

- **train_test_split** from `sklearn.model_selection` for dataset splitting.
- **pandas** as `pd` for data manipulation and handling CSV files.

A list named `feature_vector_files` was defined to store paths to four distinct CSV files. These files contain the feature vector data intended for model training and testing. The training and testing subsets were saved into new CSV files using the `to_csv()` method from pandas, with `index=False` to avoid writing row indices into the files.

B. Plotting the histograms for features

Understanding the distribution of features in training and testing datasets is crucial. Hence I plotted the histograms to understand the feature distributions of both training and testing data.

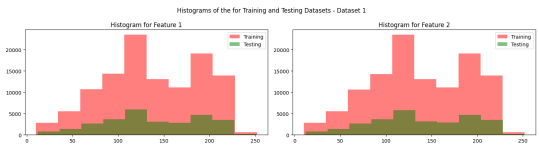

Fig. 17: Histogram of the selected features for 3 class classifier sliding data

Fig. 18: Histogram of the selected features for 3 class classifier non sliding data

Fig. 19: Histogram of the selected features for 2 class classifier sliding data

Fig. 20: Histogram of the selected features for 2 class classifier non sliding data

- I observed that for three class classifier sliding data and three class classifier non sliding data the distribution was negatively skewed. The mean (average) is located to the left of the median. This implies that the average value is pulled down by the presence of more data points of features towards the higher end.
- I observed that for two class classifier sliding data and two class classifier non sliding data the distribution was positively skewed. The mean (average) is located to the right of the median. This implies that the average value is pulled up by the presence of more data points of features towards the higher end.

C. Plotting the scatter plot for features

For visualizing feature distributions in training and testing datasets I implemented scatter plots.

- **Iterating Over Datasets:** Iterated over the indices of the dataset file paths, loading each corresponding training and testing dataset using `pd.read_csv`. This enables me to generate plots for each dataset.
- **Creating Subplots:** For each dataset, I created subplots for the training and testing data using `plt.subplots`.
- **Generating Scatter Plots:** As I move through the datasets, I generated scatter plots for both the training and testing data. This is done by utilizing `sns.scatterplot`, with '2' and '3' as the features for the x and y axes, respectively, and 'label' as the hue to differentiate data points based on their label. The `palette_color` helps color-code the labels, and I appropriately title each subplot to reflect whether it showcases training or testing data.
- **Adjusting Layout and Displaying Plots:** To minimize overlapping, I adjusted the layout of the subplots using `plt.tight_layout`. Finally, I display the scatter plots for the current dataset by executing `plt.show`.

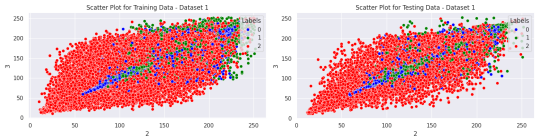

Fig. 21: Scatter plot of the selected features for 3 class classifier sliding data

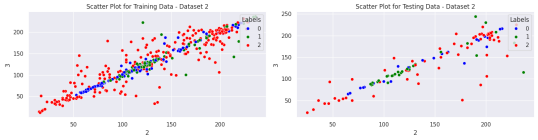

Fig. 22: Scatter plot of the selected features for 3 class classifier non sliding data

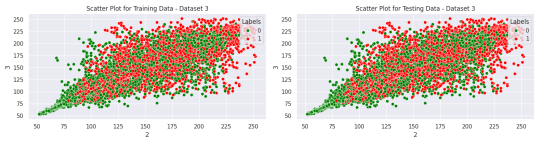

Fig. 23: Scatter plot of the selected features for 2 class classifier sliding data

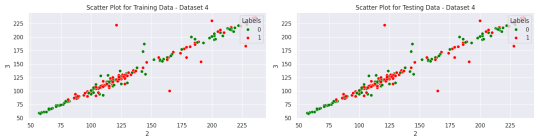

Fig. 24: Scatter plot of the selected features for 2 class classifier non sliding data

D. Task 2 (regression-based model)

Here we implement a lasso regression model and verify the accuracy and other metrics f1 score, precission and recall for each categorical file we generated. Also generated the confusion matrix.

PERFORMANCE METRICS

Metric	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Accuracy	0.5259	0.4828	0.3125	0.3984
Precision	0.5261	0.4804	0.3187	0.3508
Recall	0.5253	0.4828	0.3730	0.3999
F1 Score	0.5222	0.4669	0.2301	0.2945

CONFUSION MATRICES

Dataset 1:

3816	2445
3451	2724

Dataset 2:

19	10
20	9

Dataset 3:

0	5420	761
0	5643	519
0	13017	3319

Dataset 4:

0	21	3
0	27	4
0	49	24

E. Task 3(Random forest model)

Here we implement a Random forest model and verify the accuracy and other metrics f1 score, precission and recall for each categorical file we generated. Also generated the confusion matrix.

PERFORMANCE METRICS

Metric	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Accuracy	0.9958	0.8276	0.9958	0.8276
Precision	0.9958	0.8276	0.9958	0.8276
Recall	0.9958	0.8276	0.9958	0.8276
F1 Score	0.9958	0.8276	0.9958	0.8276

CONFUSION MATRICES

2-Class Classifier (Sliding):

6232	29
23	6152

2-Class Classifier (Non-Sliding):

24	5
5	24

3-Class Classifier (Sliding):

6232	29
23	6152

3-Class Classifier (Non-Sliding):

24	5
5	24

F. Task 4

1) Analysis of Lasso Regression Performance Metrics: Analysis of the performance metrics of a Lasso Regression model applied across four distinct datasets. The evaluation focuses on Accuracy, Precision, Recall, and F1 Score, complemented by insights from the Confusion Matrices.

- The Confusion Matrix for this Dataset 1 reveals a balanced distribution of true and false classifications. The performance metrics suggest marginal effectiveness, possibly indicating the model's struggle to distinguish between classes or a challenging dataset.
- The underperformance observed in Dataset 2, with metrics below 50%, suggests the model might not have effectively captured the underlying patterns, or the dataset features do not align well with the target variable.
- The significant drop in performance for Dataset 3, especially in F1 Score, indicates potential issues such as high dataset imbalance or overly strong regularization suppressing model learning.
- Dataset 4's metrics also denote underperformance. This might suggest non-predictive features, unsuitability of the model's assumptions for this data, or the need for model parameter adjustment.

The analysis across datasets reveals varying degrees of effectiveness by the Lasso Regression model, with no dataset showing strong performance. This suggests several potential areas for improvement, including feature engineering, adjustment of the regularization parameter, examination of data quality and balance, and the consideration of alternative models.

The overall inference is that despite Lasso Regression's known benefits for variable selection and regularization, the model requires further tuning and possibly reevaluation of its suitability for these particular datasets to enhance prediction accuracy and model fit.

2) Analysis of Random Forest Classifier Performance Metrics: Analyzed the performance metrics of a Random Forest Classifier model applied across various datasets in both "sliding" and "non-sliding" data. The evaluation focuses on key metrics such as Accuracy, Precision, Recall, and F1 Score, alongside the insights derived from Confusion Matrices.

- For sliding data confusion Matrix indicates a high level of correct classifications, with minimal false positives and negatives. These metrics suggest that the Random Forest model performs exceptionally well in distinguishing between the two classes for datasets in sliding data.
- Performance metrics in the non-sliding mode for the 3-class classifier are identical to those of the 2-class classifier in the same mode. This consistency suggests that the Random Forest model maintains its classification performance across different numbers of classes and modes, although it performs best in sliding mode scenarios.

Across different datasets and classification modes, the Random Forest Classifier showcases a robust performance, particularly in the sliding mode where it achieves near-perfect metrics. The drop in performance metrics in the non-sliding mode, though

significant, still represents a competent level of classification ability.

This analysis underscores the Random Forest model's strengths in handling both binary and multi-class classification problems, with its performance being particularly pronounced in scenarios that closely align with the sliding mode's conditions. The consistency in performance across different class configurations further attests to the model's versatility and reliability as a classification tool.

XIV. Assignment 3

A.

Constructing the PCA feature space

Feature Vector Generation for 2-Class Classification

For each class, represented by individual pictures, I generated non-overlapping feature vectors and overlapping feature vectors and assign specific captions according to the unique features of each class:

- The generated feature vectors and captions are maintained in a DataFrame named `dfnonSliding2class_pca`.
- PCA is then applied to this DataFrame using the `Apply_pca` function.
- Post-PCA, both non-overlapping and overlapping feature vectors for each class are further processed, captioned, and exported to a CSV file. The final data is stored in the `dfsliding2class_pca` DataFrame.

Feature Vector Generation for 3-Class Classification

For each class, represented by individual pictures, I generated non-overlapping feature vectors and overlapping feature vectors and assign specific captions according to the unique features of each class:

- The generated feature vectors and captions are maintained in a DataFrame named `dfnonSliding3class_pca`.
- PCA is then applied to this DataFrame using the `Apply_pca` function.
- Post-PCA, both non-overlapping and overlapping feature vectors for each class are further processed, captioned, and exported to a CSV file. The final data is stored in the `dfsliding3class_pca` DataFrame.

After generating the feature vectors and applying PCA to the previous dataframes, we then split the feature vectors into training and testing sets. This allows us to apply our models to the training data and later evaluate their performance on the test data.

B. Task 3 Lasso Regression

Here we implement a lasso regression model for the feature space obtained after applying the PCA and verify the accuracy and other metrics f1 score, precission and recall for each categorical file we generated. Also generated the confusion matrix. Below are the values for lasso regression. **Results and Comparision Dataset 1**

- Accuracy decreased from 0.5259 to 0.5101.
- Precision decreased from 0.5261 to 0.5098.
- Recall decreased from 0.5253 to 0.5096.
- F1 Score decreased from 0.5222 to 0.5079.

TABLE I: Lasso Model Performance Metrics

Dataset	Accuracy	Precision	Recall	F1 Score	Conf. Matrix
1	0.5101	0.5098	0.5096	0.5079	3578 2683 3410 2765
2	0.5345	0.5366	0.5345	0.5276	19 10 17 12
3	0.3279	0.3211	0.3793	0.2439	0 5324 857 0 5564 598 0 12496 3840
4	0.3672	0.3536	0.3878	0.2689	0 22 2 0 28 3 0 54 19

Conclusion: Slight decrease in performance indicates potential loss of useful information due to PCA.

Dataset 2

- Accuracy increased from 0.4828 to 0.5345.
- Precision increased from 0.4804 to 0.5366.
- Recall remained approximately the same.
- F1 Score increased from 0.4669 to 0.5276.

Conclusion: Improvement in performance suggests that PCA helped by removing noise or irrelevant features.

Dataset 3

- Accuracy increased from 0.3125 to 0.3279.
- Precision increased from 0.3187 to 0.3211.
- Recall increased from 0.3730 to 0.3793.
- F1 Score increased from 0.2301 to 0.2439.

Conclusion: Marginal improvement indicates that PCA concentrated useful variance into fewer components.

Dataset 4

- Accuracy decreased from 0.3984 to 0.3672.
- Precision decreased from 0.3508 to 0.3536.
- Recall decreased slightly from 0.3999 to 0.3878.
- F1 Score decreased from 0.2945 to 0.2689.

Conclusion: Decrease in most metrics post-PCA, suggesting potential removal of critical data nuances.

Overall Findings PCA has a varied impact on the datasets. It can improve model efficiency by eliminating noise and redundant features in some cases, whereas it may reduce necessary information critical for effective model performance in others. Dataset characteristics should be considered when applying PCA.

C. Random Forest

Here I implemented a Random forest model for the feature space after applying the pca and verified the accuracy and other metrics f1 score, precission and recall for each categorical file we generated. Also generated the confusion matrix.

TABLE II: Random Forest Classifier Results

Dataset	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
1	0.9565	0.9567	0.9564	0.9565	6054 207 334 5841
2	0.7069	0.7132	0.7069	0.7047	23 6 11 18
3	0.9573	0.9577	0.9572	0.9573	6076 185 346 5829
4	0.7069	0.7132	0.7069	0.7047	23 6 11 18

Conclusion Across all datasets, there is a decrease in all performance metrics (accuracy, precision, recall, and F1 score) after applying PCA. There will be a decrease in accuracy but it is important to improve the performance.

D. Plotting the scatter plots for features

For visualizing feature distributions in training and testing datasets I implemented scatter plots.

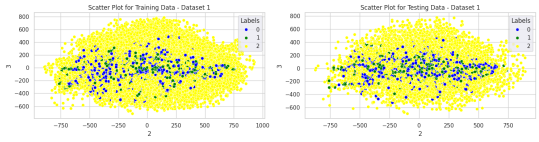

Fig. 25: Scatter plot of the selected features for 3 class classifier sliding data

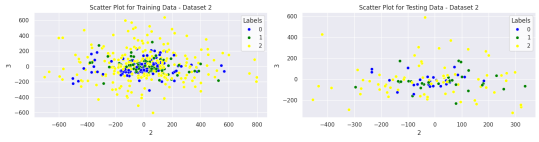

Fig. 26: Scatter plot of the selected features for 3 class classifier non sliding data

Fig. 27: Scatter plot

E. Plotting the Histograms for the features

Understanding the distribution of features in training and testing datasets is crucial. Hence I plotted the histograms to understand the feature distributions of both training and testing and data.

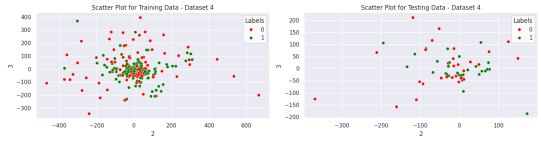

Fig. 28: Scatter plot

Fig. 29: Histogram of the selected features for 3 class classifier sliding data

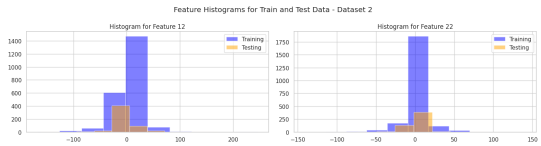

Fig. 30: Histogram of the selected features for 3 class classifier non sliding data

Fig. 31: Histogram of the selected features for 2 class classifier non sliding data

Fig. 32: Histogram of the selected features for 2 class classifier non sliding data