Software Engineering Lab Spring 2024

Assignment-4: NumPy and its Applications

Name :- B. Sri Chaitanya

Roll No:- 22CS10018

Learning objectives: NumPy stands for Numerical Python. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

1. Write a program to load a .csv file as a NumPy 1-D array. Find the maximum and minimum elements in the array.

A)
```
import numpy as np
```

```
# The file variable

csv_file = 'book1.csv'



# Importing the of the 2nd column, i.e the numbers into a numpy 1D-
array

# Skipping column 1 and row 1 of column 2 to access the require data

data = np.loadtxt(csv_file, delimiter = '\t', usecols = 1, skiprows=1)



# Maximum element

maximum = np.max(data)



# Minimum element

minimum = np.min(data)



# Printing the maximum and minimum elements

print("The maximum element is :- {}\nThe minimum element is :-
{}".format(maximum, minimum))
```

2 For the Numpy 1-D array as obtained in Q.1, sort the elements in ascending order.

A)

```
import numpy as np
```

```
# The file variable

csv_file = 'book1.csv'



# Importing the of the 2nd column, i.e the numbers into a numpy 1D-
array

# Skipping column 1 and row 1 of column 2 to access the require data

data = np.loadtxt(csv_file, delimiter= '\t', usecols = 1, skiprows= 1)



# Sorting the array in ascending order

# We do not data.sort() so that the original data isn;t modified

sorted_data = np.sort(data)



# Printing the sorted data

print("The sorted data is :-\n", sorted_data )
```

3. For the sorted Numpy 1-D array as obtained in Q.2, reverse the array and print.

A)import numpy as np

```
# The file variable

csv_file = 'book1.csv'
```

```
# Importing the of the 2nd column, i.e the numbers into a numpy 1D-
array

# Skipping column 1 and row 1 of column 2 to access the require data

data = np.loadtxt(csv_file, delimiter= '\t', usecols = 1, skiprows= 1)



# Sorting the array in ascending order

# We do not data.sort() so that the original data isn;t modified

sorted_data = np.sort(data)



# Reversing the sorted data array

sorted_data = sorted_data[::-1]



# Printing the sorted data

print("The sorted and reversed data is :-\n", sorted_data )
```

4. Write a program to load three .csv files (Book1.csv, Book2.csv, and Book3.csv) as a list of Numpy 1-D arrays. Print the means of all arrays as a list.

A)

```
import numpy as np



# The list of all file names

csv_file = ['book1.csv', 'book2.csv', 'book3.csv']



# List to store the numpy arrays
```

```
all_data = []



# Iterating through all the files

for x in csv_file:

    # Importing the of the 2nd column, i.e the numbers into a numpy 1D-
array

    # Skipping column 1 and row 1 of column 2 to access the require
data

    data = np.loadtxt(x, delimiter='\t', usecols= 1, skiprows= 1)



    # Appending the numpy array into the (global) list

    all_data.append(data)



# Creating a list of means for each numpy array in all_data

mean = [np.mean(np.array(x)) for x in all_data]



# Printing the means

print("The mean of all arrays is:-", mean, sep= ' ')
```

5. Write a program to read an image, store the image in NumPy 3-D array. For the image, consider a.PNG. Display the image. Let the image stored in the NumPy array be X.

A)

```
import cv2

import numpy as np
```

```
# Reading the image
image_path = 'a.png'  # Path to the image

# Reading the image in BGR format ( in color)
img = cv2.imread(image_path, cv2.IMREAD_COLOR)

# Displaying the image in a window called "Image"
cv2.imshow('Image', img)

# Wait till any key is pressed
cv2.waitKey(0)

# Closing all OpenCV windows
cv2.destroyAllWindows()

# Converting the BGR format into RGB format for matplotlib.pylot
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Storing the image now in a 3-D numpy array (X)
X = img_rgb
```

6.Write a program to convert a color image (say a.PNG) into a grescale image. Let the greysacle image stored in the Numpy 2-D array be X. Display the grayscale iamge on the screen.

A)

```
import cv2

import numpy as np



# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Since OpenCV's imshow expects a 2D array for a grayscale image, we
can directly use X

cv2.imshow('Grayscale Image by Mean of RGB Values', X)



# Wait for a key press and then close all OpenCV windows

cv2.waitKey(0)

cv2.destroyAllWindows()
```

7. Let Y be the transpose matrix of X. Write a program to obtain Z = X×Y.

A)

```
import cv2

import numpy as np



# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Computes the transpose of X

Y = X.T



# Multiplying X with Y

Z = np.dot(X,Y)



# Printing the matrices
```

```
print("The matrix X is :- \n{}\nThe matrix Y is :-\n{}\nThe matrix Z is
:-\n{}\n".format(X,Y,Z))
```

8. For the problem in Q. 7, write your program without using NumPy library. Compare the computation times doing the same with NumPy and basic programming in Python.

A)

```
import cv2

import numpy as np

import time



# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Storing the start time

start_time = time.time()
```

```
# Calculating Z the numpy way


# Computing the transpose of X
Y = X.T


# Calculating the product XY
Z = np.dot(X,Y)


# End of numpy time
numpy_time = time.time()


# Priting the time required to calcute Z using numpy
print("The time to perform the multiplication using numpy is :- ",
numpy_time - start_time, "s")


# Converting the nunmpy array X to a list X_list
X_list = X.tolist()


# Start time for matrix style multiplication
matrix_time = time.time()


Y_list = [list(row) for row in zip(*X_list)]


# Perform matrix multiplication between X_list and Y_list to get Z_list
```

```
# Initialize Z_list with the appropriate dimensions filled with zeros
Z_list = [[0 for _ in range(len(X_list))] for _ in range(len(X_list))]




# Populate Z_list with the results of the matrix multiplication
for i in range(len(X_list)):
    for j in range(len(X_list)):
        for k in range(len(Y_list[0])):
            Z_list[i][j] += X_list[i][k] * Y_list[k][j]


# End time for matrix style multiplication
end_time = time.time()


# Printing the caluclation time
print("The time taken ot multiply normally is :- ", end_time -
matrix_time, "s")
```

9. Let Y be the transpose matrix of X. Write a program to obtain Z = X×Y. For the problem in Q. 7, write your program without using NumPy library. Compare the computation times doing the same with NumPy and basic programming in Python. Plot the pixel intensity histogram of the greyscale image stored in X.

A)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Calculate the histogram of pixel intensities in the grayscale image

histogram, bin_edges = np.histogram(X, bins=256, range=(0, 256))



# Plot the histogram

plt.figure("Plot")

plt.title("Grayscale Histogram")

plt.xlabel("Pixel Intensity")

plt.ylabel("Pixel Count")

plt.xlim([0, 256])  # Ensure the x-axis covers the full range of pixel
values

plt.plot(bin_edges[:-1], histogram)  # Plot the histogram

plt.show()
```

10. Create a black rectangle at the position [(40,100) top right, (70, 200) bottom left] in the grayscale image. Display the image.

A)

```
import cv2

import numpy as np



# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Create a black rectangle on the image

cv2.rectangle(X, (40, 100), (70, 200), (0), thickness=-1)



# Display the image with the rectangle

cv2.imshow('Grayscale Image with Black Rectangle', X)

cv2.waitKey(0)  # Wait for a key press to close the window

cv2.destroyAllWindows()
```

11. Using the grayscale image stored in X, transform it into the binarized image with thresholds: [50, 70, 100, 150]. Let the binarized images are stored in Z50, Z70, Z100, and Z150, respectively.

A)

```
import cv2

import numpy as np



# Read the image

image_path = 'a.png'  # Specify the path to your image

img = cv2.imread(image_path)



# Converting the image from BGR to RGB

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



# Calculating the mean of the RGB values for each pixel to get the
grayscale image

X = np.mean(img_rgb, axis=-1).astype(np.uint8)



# Binarize the image for each threshold

thresholds = [50, 70, 100, 150]

Z50 = (X > thresholds[0]).astype(np.uint8) * 255

Z70 = (X > thresholds[1]).astype(np.uint8) * 255

Z100 = (X > thresholds[2]).astype(np.uint8) * 255

Z150 = (X > thresholds[3]).astype(np.uint8) * 255
```

```
# Displaying the images using opencv
cv2.imshow('Original Grayscale Image', X)
cv2.imshow('Binarized Image Z50', Z50)
cv2.imshow('Binarized Image Z70', Z70)
cv2.imshow('Binarized Image Z100', Z100)
cv2.imshow('Binarized Image Z150', Z150)

# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows() ## Destroy all windows
```

12. Consider the color image stored in a.png. Create a filter of [[-1,-1,-1][0,0,0][1,1,1]], and multiply this filter to each pixel value in the image. Display the image after filtering.

A)

```
import cv2
import numpy as np

# Read the image
image_path = 'a.png'  # Specify the path of my  image
img = cv2.imread(image_path)

# Define the filter (kernel)
kernel = np.array([[-1, -1, -1],
```

```
                    [ 0,  0,  0],
                    [ 1,  1,  1]])


# Apply the filter to the image using cv2.filter2D
filtered_img = cv2.filter2D(img, -1, kernel)


# Display the original and filtered images
cv2.imshow('Original Image', img)
cv2.imshow('Filtered Image', filtered_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

==========

==========

==========XXXXXX==========