Machine Learning Assignment-1

Team-5

Team members:

1. Sanjana. R – BL.EN.U4AIE22149

2. Aditi Ajay Marar- BL.EN.U4AIE22103

3. G. Krisha – BL.EN.U4AIE22114

Set-1: Odd number set.

R. Sanjana – BL.EN.U4AIE22149

1. Finding pairs of numbers from list suck that their sum is 10. A list is taken as input in the main function and sent to this function.

Algorithm:

```
FUNCTION find_pairs_of_ten(input_list)
        pairs_of_sum10 <- emptylist
        FOR i FROM 0 to length(input_list)-1 DO
                Current_value <- input_list[i]
                FOR j FROM 0 TO length(input_list)-1 DO
                        IF input_list[j]+current_value EQUALS 10 THEN
                                Add (current_value, input_list[j]) TO pairs_of_sum10
                        END IF
                END FOR
        END FOR
        RETURN pairs_of_sum10
END FUNCTION
```

2. To find the range of a list, we need to find the largest and smallest value in the list and then find their difference.

To find the smallest element:

```
FUNCTION minimum(input_list)
        min <- 100000 #A large value to compare
        FOR I FROM 0 TO length(input_list)-1 DO
                IF input_list[i] LESS THAN min THEN
                        min <- input_list[i]
                END IF
        END FOR
        RETURN min
END FUNCTION
```

To find the largest element:

```
FUNCTION maximum(input_list)
        max <- -100000 #Small value to compare
        FOR i FROM 0 TO length(input_list)-1 DO
                IF input_list[i] GREATER THAN max THEN
                        max <- input_list[i]
                END IF
        END FOR
        RETURN max
END FUNCTION
```

find_list_range() function is called by the main function to find the range of a list. It uses the functions minimum() and maximum() in its execution. If the length of the list is lesser than 3, range determination is not possible.

```
FUNCTION find_list_range(input_list)
        IF length(input_list) LESS THAN 3 THEN
```

```
            RETURN None
        END IF
        min_value <- minimum(input_list)
        max_value <- maximum(input_list)
        range <- max_value - min_value
        RETURN range
END FUNCTION
```

3. To find A^m, a square matrix is take as input from the user in the main(), and find_matrix_power() is called, which in turn uses 2 functions create_matrix() and multiply_matrices(). create_matrix() is used to create a zero matrix of the size of the matrix given as argument to it. mulitply_matrices() takes in 2 matrices and returns their product.

```
FUNCTION create_matrix(matrix)
        matrix_2 <- emptylist
        FOR I FROM 0 TO length(matrix)-1 DO
            row <- emptylist
            FOR j FROM 0 TO length(matrix[0])-1 DO
                ADD 0 TO row
            END FOR
        ADD row to matrix_2
        END FOR
        RETURN matrix_2
END FUNCTION
```

```
FUNCTION multiply_matrices(matrix1, matrix2)
        result_matrix <- create_matrix(matrix1)
        FOR i FROM 0 to length(matrix1)-1 DO
            FOR j FROM 0 TO length(matrix2[0])-1 DO
                FOR k FROM 0 TO length(matrix2)-1 DO
                    result_matrix[i][j] <- result_matrix[i][j]+matrix1[i][k]*matrix2[k][j]
```

```
                    END FOR
                END FOR
            END FOR
            RETURN result_matrix
END FUNCTION
```

find_matrix_power() does multiplication of given matrix with itself to 'power-value' times to get A^m.

```
FUNCTION find_matrix_power(matrix,power_value)
        original_matrix <- COPY of 'matrix'
        FOR i FROM 0 TO power_value-1 DO
                matrix <- multiply_matrices(matrix,original_matrix)
        END FOR
        RETURN matrix
END FUNCTION
```

4. To find the most occurring letter in a given string we first create a dictionary of all letter in the string and later add the count of each letter to it and then find the maximum one out of it.

```
FUNCTION find_max_occuring_letter(input_string)
        char_list <- emptylist
        FOR letter in input_string DO
                IF letter NOT IN char_list
                        Add letter to char_list
                END IF
        END FOR
        FOR char in char_list
                Add char:0 to char_count
        END FOR
        FOR current_letter in input_string DO
                count <- 0
                FOR compare_letter in input_string DO
```

IF compare_letter EQUALS current_letter THEN

count <- count+1

END IF

END FOR

char_count[current_letter] <- count

END FOR

max_value <- maximum value of char_count.values()

most_occuring_char <- char with maximum count value

RETURN (most-occuring_char, max_value)

END FUNCTION

All the results retruned by each function is printed to user in main function.

Aditi Ajay Marar – BL.EN.U4AIE22103

Q1) Consider the given list as [2,7,4,1,3,6]. Write a program to count pairs of elements with a sum equal to 10

The pair_10 function looks for pairs of numbers in a given list that add up to 10. It checks each number in the list and finds a value, when added to the no. give a sum of 10. If the no. is also in the list, it adds the pair to the result. If the number 5 appears exactly once, it excludes the pair [5, 5] from the final result. The function returns the count of pairs and a list of those pairs.

Function pair_10 with argument as list_of no:

Initialize variables list_containing_pairs=[]

For variable in list_of_no

Initialize pair_ten = 10-variable

If pair_ten *in* list_of_no

If list [pair_ten,variable] not in list_containing_pairs

Append [variable, pair_ten] *in* list_containing_pairs

If count of 5 in list_of_no ==1

Remove list [5,5] from list_containing_pairs

Initialize result=[length of list_containing pairs, list_containing_pairs]

Return result

Print QUESTION 1~CONSIDER THE GIVEN LIST [2,7,4,1,3,6] WAP TO COUNT PAIRS OF ELEMENTS WITH SUM = 10

Input input_str

Initialize input_list by splitting input_str using split function with argument ","

For index_q1 in range of length of input_list

    Convert the elements to int

Call function pair_10 with argument input_list and save the result in result_of_q1

If result_of_q1[0] == 0

    Print No pairs in the list that sum up to 10

Else

    Print] result_of_q1[0] and result_of_q1[1]

Q2) Write a program that takes a list of real numbers as input and returns the range (difference between minimum and maximum) of the list. Check for list being less than 3 elements in which case return an error message (Ex: "Range determination not possible"). Given a list [5,3,8,1,0,4], the range is 8 (8-0)

The max_min_range function calculates the range of a list of numbers by finding the difference between its maximum and minimum values. If the list has less than three elements, it returns an error message. It iterates through the list to identify the maximum and minimum values and returns it in a string.

Function max_min_range with argument list_of_no

    If length of list_of_no < 3

        Print Error message

    Else

        Initialize both maximum and minimum as list_of_no[0]

        For index in range of 1 to length of list_of_no

            If list_of_no[index]>maximum

                Maximum=list_of_no[index]

            If list_of_no[index]<minimum

                Minimum=list_of_no[index]

Initialize result = “Range of the given matrix is (”+maimum+”,”+minimum+”)”

Return result

Input input_str2

Initialize input_list2 by splitting input_str2 using split function with argument “,”

For index_q2 in range of length of input_list2

Convert the elements to int

Call function max_min_range with argument input_list2 and save the result in result_of_q2

Print the result

Q3) Write a program that accepts a square matrix A and a positive integer m as arguments and returns A^m.

The matrix_multiply function multiplies two square matrices, 'a' and 'b'. The function first checks the length of one of the matrices (Note: we are only multiplying square matrices). It then iterates through each row and column of the matrices using nested loops. Within the loops, the function calculates each element of the resulting matrix by multiplying corresponding elements from the row of matrix 'a' and column of matrix 'b' and gives the results. The computed elements are stored in a new matrix, 'result_matrix,' which is then returned.

Function matrix_multiply with arguments a and b

Initialize result_matrix=[] and length as length of a

For i in range 0 to length

Initialize row =[]

For j in range 0 to length

Initialize variable=0

For k in range 0 to length

Variable += a[i][k]*b[k][j]

Append variable in row

Append row in result_matrix

Return result_matrix

The matrix_power function calculates the power of a square matrix using repeated matrix multiplication. It takes two arguments: const(power), and matrix(a square matrix). If the specified power is greater than 0, the function initializes a result matrix with the input matrix and then

iteratively multiplies it by the original matrix const - 1 times. The final result is matrix to the power const. If the given power is not greater than 0, the function returns -1, indicating an error.

```
Function matrix_power with arguments const and matrix
    If const>0
        Initialize result_matrix=matrix
        For i in range 1 to const
            Call function matrix_multiply with arguments result_matrix and save it in variable matrix
        Return result_matrix
    Else
        return -1

Print QUESTION 3~ WAP that accept a a square matrix A and a +ve no. m and return A^m
Input row/column of matrix as r_c
Input +ve integer constant k
Initialize input_matrix_q3=[]
for i in range 0 to r_c
    initialize row=[]
    for j in range 0 to r_c
        Input var as the variable of the matrix at position i,j
        Append var in row
    Append row in input_matrix_q3
Call function matrix_power with arguments k, input_matrix_q3 and save it in variable ans_3
If -1==ans_3
    Print Error message
Else
    For i in ans_3
        Initialize output_str_q3=""
        For j in i
            output_str_q3+= str(j)+" "
        Print output_str_q3
```

Q4)

Write a program to count the highest occurring character & its occurrence count in an input string. Consider only alphabets. Ex: for “hippopotamus” as input string, the maximally occurring character is ‘p’ & occurrence count is 3

The highest_char function is used to find the character with the highest frequency of the string given by user. It starts by checking if the string contains only alphabets using the isalpha(). If true, the function creates a dictionary, dict_for_characters_n_count, to store the frequency of each character in the string with the help of the for loop. It finds the character with the highest frequency by iterating through the dictionary. Finally, the function returns a list containing: the maximum frequency and the corresponding character. If the string also contains no.s and special characters it returns [-1, ""]

```
Function highest_char with argument string
    If string.isalpha()
        Initialize dict_for_characters_n_count={}
        For character in string
            If character not in dict_for_characters_n_count
                dict_for_characters_n_count[character]+=1
            Else
                dict_for_characters_n_count[character]+=1
        Initialize maximum=0 and max_char=””
        For temp in dict_for_characters_n_count
            If dict_for_characters_n_count[temp]>maximum:
                maximum=dict_for_characters_n_count[temp]
                max_char=temp
        Initialize result=[maximum,max_char]
        Return result
    Else
        Return[-1,””]

Print WAP TO COUNT THE HIGHEST OCCURING CHARACTER
```

Input a word and save it in inp_str

Call function highest_char with argument inp_str save it in result_of_q4

If result_of_q4[0]==-1*:*

Print Invalid input

Else

Print the result_of_q4[0] and result_of_q4[1]

Set-2: Even number set

G. Krisha – BL.EN.U4AIE22114

For function vowels_and_consonants

Input is given as string of characters

Output will be shown as integers that tells the count of vowel and consonant

1) vowels <-- ”AEIOUaeiou”
2) vowel_count<-- 0
3) Consonants_count<--0
4) For every character in the string:
    a. if character is a alphabet:
        1. if character is in vowels:
            then increment vowel_count
        2. else:
            then increment consonant_count
5) return vowel_count and consonant_count

For function matrix_multiplication

Input is given as two matrices matrix_1 and matrix_2

Output is shown as if possible for multiplication the multiplied result matrix is printed else return an string “Error”

1) if the number of columns in matrix_1 is not equal to number of rows:
Then return string “Error”

2) result_matrix <-- empty list
3) For every row in matrix_1:
    a. row_value <-- empty list
    b. For each column in matrix_2[0]:
        1. element<--0
        2. for every index in range of length of matrix_2):
            Do matrix_1[row][index] * matrix_2[row][index]

            Increment element

        3. append element to row_value
    c. Append row value to result_matrix
4) return the result_matrix

For function common_elements_in_list

Input is given as two lists list_1 and list_2

Output is shown as count of common_elements as integers and common_elements as list

There are two functions

1) string_to_integer
2) Common_elements_in_the_list

For the function string_to_integer

1) split input_str by commas to get result_list
2) For every element in result_list:
    convert element to integer lists

3) Return result_list

For the function common_elements_in_the_list

1) call the function string_to_integer to convert the two list list_1 and list_2 to integer list
2) common_elements<-- empty list
3) For every variable in list_!:
    a. If variable in list_2:
        then append variable to common_elements

4) return count of common_elements and common_elements

For function transpose_of_the_matrix

Input is given as matrix input_matrix in the main function

Output is given as transpose of the matrix result_matrix

1) result_matrix <-- empty list
2) For every column in matrix[0]:
    a. Row <-- empty list
    b. For every row in matrix:
        append matrix[row][column] to row
    c. Append row to result_matrix
3) Return result_matrix

For main function

For consonants and vowels

1) string_input takes input from user and stores the value
2) Call the function vowels_and_consonants with argument string:
    return values consonant_count and vowel_count
3) Print result

For matrix_multiplication

1) input_matrix1_row from user
2) input_matrix1_column from user
3) input_matrix2_row from user
4) input_matrix2_column from user
5) matrix1_input s<-- empty list
    a. rows_values <-- empty list
    b. for each column in range matrix2_column:
        1. user input for matrix1 for entering of elements
        2. Convert input to integer and store in elements1
        3. Append elements1 to row_values
    c. append row_values to matrix1_inputs
6) print matrix1_inputs
7) matrix2_inputs <-- empty list
8) For row in range input_matrix2_row:
    a. rows_values <--empty list
    b. for each column in range matrix1_column:
        1. User input for matrix2 for entering of elements
        2. Convert input to integer and store in elements2
        3. Append elements2 to rows_value
    c. append rows_value to matrix2_inputs

5) Print matrix2_inputs
6) Call the function matrix_multiplication with matrix1_inputs and matrix2_inputs:
   return multiplied matrix

7) Print multiplied matrix

For common_elements_in_the_list

1) list_1 input from user
2) list_2 input from user
3) Call function common_elements_in_the_list with arguments list_1 and list_2:
   return count of common_elements and common_elements

4) Print common_elements and count of common_elements

For function transpose_of_the_matrix

1) input_matrix_row from user
2) input_matrix_column from user
3) matrix_input <-- empty set
4) For every row in range input_matrix_row:
   a. rows<--empty set
   b. For every column in range input_matrix_column:
      1. User input for entering elements
      2. Convert input to integer and store in element
      3. Append element to rows
   c. append rows to matrix_input
5) print input matrix
6) call function transpose_of_the_matrix:
   return transposed matrix

7) print transposed matrix