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 Am, 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 Am.

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 inlist\_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:
5. if character is a alphabet:
6. if character is in vowels:

then increment vowel\_count

1. else:

then increment consonant\_count

1. 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”

1. result\_matrix <-- empty list
2. For every row in matrix\_1:
3. row\_value <-- empty list
4. For each column in matrix\_2[0]:
5. element<--0
6. for every index in range of length of matrix\_2):

Do matrix\_1[row][index] \* matrix\_2[row][index]

Increment element

1. append element to row\_value
2. Append row value to result\_matrix
3. 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

1. 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\_!:
4. If variable in list\_2:

then append variable to common\_elements

1. 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]:
3. Row <-- empty list
4. For every row in matrix:

append matrix[row][column] to row

1. Append row to result\_matrix
2. 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

1. 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
6. rows\_values <-- empty list
7. for each column in range matrix2\_column:
8. user input for matrix1 for entering of elements
9. Convert input to integer and store in elements1
10. Append elements1 to row\_values
11. append row\_values to matrix1\_inputs
12. print matrix1\_inputs
13. matrix2\_inputs <-- empty list
14. For row in range input\_matrix2\_row:
15. rows\_values <--empty list
16. for each column in range matrix1\_column:
17. User input for matrix2 for entering of elements
18. Convert input to integer and store in elements2
19. Append elements2 to rows\_value
20. append rows\_value to matrix2\_inputs
21. Print matrix2\_inputs
22. Call the function matrix\_multiplication with matrix1\_inputs and matrix2\_inputs:

return multiplied matrix

1. 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

1. 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:
5. rows<--empty set
6. For every column in range input\_matrix\_column:
7. User input for entering elements
8. Convert input to integer and store in element
9. Append element to rows
10. append rows to matrix\_input
11. print input matrix
12. call function transpose\_of\_the\_matrix:

return transposed matrix

1. print transposed matrix