Machine Learning Assignment-2

Team-5:

Team members:

1. Sanjana. R – BL.EN.U4AIE22149

2. Aditi Ajay Marar- BL.EN.U4AIE22103

3. G. Krisha – BL.EN.U4AIE22114

Sanjana. R – BL.EN.U4AIE22149

1) a. Euclidean distance

It finds the Euclidean distance of 2 vectors by using a for loop to take elements with same index in both vectors, find their difference, square it and add it to a variable. The square root of the variable is returned as the output.

```
FUNCTION Euclidean_distance(vector1, vector2)
    sum_square_of_differences <- 0
    FOR i FROM 0 TO length(vector1)-1 DO
            X <- vector1[i]
            Y <- vector2[i]
            sum_square_of_differences= sum_square_of_differences+square(Y-X)
    END FOR
    RETURN square_root(sum_square_of_differences)
END FUNCTION
```

2) a. Manhattan distance

It find the Manhattan distance of 2 vectors. It uses a for loop and finds the difference of corresponding elements of the 2 vectors. The differences of each pair is added and returned as output.

```
FUNCTION Manhattan_distance(vector1, vector2)
        Dist <- 0
        FOR i FROM 0 TO length(vector1)-1 DO
                X <- vector1[i]
                Y <- vector1[i]
                Dist <- Dist+(Y-X)
        END FOR
        RETURN Dist
END FUNCTION
```

3) K-nn classifier

The Iris dataset is taken as input and is used to train the k-nn model. The dataset is read and split into lines. Each line except the header line is read and split into words. The label column is ignored.

The Euclidean distance between the input vector containing the features and each record in the dataset (excluding the label column) is found. The k-nearest neighbours is found by sorting the Euclidean distances. The corresponding labels of the indices are found and a count is taken. The label with highest count is the final prediction.

```
FUNCTION Knn(input_vector, k)
    OPEN dataset
    READ dataset
    SPLIT dataset INTO lines
    Distances <- EMPTY_DICTIONARY
    Dist <- EMPTY_LIST
    Classes_count <- Dictionary of classes of dataset: 0 #Initial count is 0.
    FOR line in LINES DO
        Vector_1 <- SPLIT line INTO words (by spaces)
        Dist_curr_vector <- Euclidean_distance(Vector_1, input_vector)
        ADD Dist_curr_vector to Dist
    END FOR
    FOR index FROM 0 TO length(Dist)-1 DO
        ADD index:Dist[index] TO Distances
    END FOR
    distances_sorted <- SORT Distances based on distance_value
    sorted_indices <- Sorted keys of the Distances dictionary
    top_matches <- EMPTY_LIST
    FOR i FROM 0 TO k DO
        ADD (sorted_indices[i], distances_sorted[i]) TO top_matches
    END FOR
    Class_labels <- corresponding labels of the top_matches
    FOR i in class_lablels DO
        ADD 1 to count of corresponding label in classes_counts
    END FOR
```

```
        FIND THE MAXIMUM OCCURING LABEL AND RETURN IT
    FUNCTION END
```

4) Label Encoding

The Iris dataset is used. The dataset is read as lines excluding the header line. Through traversal unique classes are found and put into a list. A number is assigned to each class by using a iterative variable ‘j’. The encoding is returned as the output. By changing the dataset, any number of labels can be encoded as there is no hardcoding done for the encoding.

```
    FUNCTION Label_Encoding()
        OPEN dataset
        READ dataset
        SPLIT dataset INTO lines
        classes <- EMPTY_LIST
        FOR line FROM 2nd LINE DO
            field <- SPLIT line INTO words
            label <- field [-1]
            IF label IS NOT IN classes THEN
                ADD label TO classes
            END IF
        END FOR
        class_encoding <- EMPTY_DICTIONARY
        FOR j FROM 0 TO length(classes)-1 DO
            ADD classes[j]:j TO class_encoding
        END FOR
        RETURN class_encoding
    END FUNCTION
```

5) One hot Encoding

The Iris dataset is used. The dataset is traversed and unique classes are put into a list. A number is assigned to each unique class. The maximum possible digits in the binary version is found. The number assigned to a class is converted to binary, the letter ‘b’ which is generated by python is removed and the binary value is appended with sufficient 0’s in appropriate places to maintain same length of encoding. The encoding can be extended to any number of labels.

```
    FUNCTION One_hot_encoding()
```

```
        OPEN dataset
        READ dataset
        SPLIT dataset INTO lines
        classes <- EMPTY_LIST
        FOR line FROM 2nd LINE DO
            Field <- SPLIT line INTO words
            Label <- Field[-1]
            IF Label IS NOT IN classes THEN
                ADD Label TO classes
            END IF
        END FOR
        max_len <- length of binary(length of classes)-1)
        encoding <- EMPTY_LIST
        FOR j FROM 0 TO length(classes)-1 DO
            ADD classes[j]: string(binary(j)) TO encoding
        END FOR
        RETURN encoding
    END FUNCTION
```

Aditi Ajay Marar – BL.EN.U4AIE22103

Q1 EUCLIDEAN AND MANHATTAN

```
FUNCTION Manhattan, with x and y as arguments
    INITIALIZE m_l as an empty list
    FOR i in range 0 to length of list x
        INITIALIZE var as x[i]-y[i]
        IF var is negative then multiply var with -1
        APPEND var in m_l
    INITIALIZE m_val=0
    FOR i in m_l
        INCREMENT m_val by i
    RETURN (m_val,m_l)
```

```
FUNCTION Euclidean with arguments as x and y
    CALL manhattan with arguments x and y as save it as m
    INITIALIZE e_val as 0
    FOR i in range 0 to m[1]:INCREMENT e_val by square of i
    INITIALIZE ans as square root of e_val
    RETURN ans
```

Q2 KNN CLASSIFIER

It has 3 functions one to get the frequency of which type has the highest count, sorting it wrt the distance, and lastly to receive the answer itself

```
FUNCTION freq with argument l
    INITIALIZE an empty dictionary for f
    For I in l
        IF i not in f then f[i]=1
        ELSE, f[i]+=1
    INITIALIZE m as 0 and v as ‘’
    FOR i in f
        IF f[i] is greater than m then,
        m=f[i]
        v=i
    RETURN (v,m)
```

```
FUNCTION sortknn with arguments d, val, x, and y
    INITIALIZE n as length of d
    FOR i in range 0 to n-1
        FOR j in range 0 to n-1-i
```

```
            IF d[j]>d[j+1], then
                d[j], d[j+1] = d[j+1], d[j]
                x[j], x[j+1] = x[j+1], x[j]
                y[j], y[j+1] = y[j+1], y[j]
                val[j], val[j+1] = val[j+1], val[j]
    INITIALIZE ans as [[x[i], y[i], d[i], val[i]] for i in range(n)]
    RETURN ans


FUNCTION knn with arguments X,Y,x,y,k,val
    INITIALIZE dist as []
    for i in range 0 to len(x): dist+=[math.sqrt(((x[i]-X)**2)+((y[i]-Y)**2))]
    IF k is greater than length of x then RETURN -1, this is to show error incase k value is bigger than the
    entries given
    ELSE,
        CALL function sortknn with arguments dist,val,x,y save it as o
        INITIALIZE ans as []
        FOR i in range 0 to k: ans+= [o[i][3]]
        CALL function freq with argument ans  and save it in final
        RETURN [final[0], o]
```

Q3 & 4 LABEL ENCODING AND ONE HOT ENCODING

```
FUNCTION label_encoding with argument label
    INITIALIZE d as []
    FOR i in label
        IF i not in d then append I in d
    INITIALIZE decoded as [d.index(i) for i in label]
    RETURN  (decoded, d)


FUNCTION one_hot with argument label
CALL function label_encoding with argument label and save it in l
    INITIALIZE inp as l[0], d as l[1] and decoded as []
```

```
FOR i in range 0 to length of inp:
        INITIALIZE o as [0 for i in range(len(d))]
        o[inp[i]]=1
        append o in decoded
RETURN decoded
```

G. Krisha – BL.EN.U4AIE22114

For function euclidean_distance

```
Function euclidean_distance(v1,v2):
    1)  dist<--0
    2)  For each I in length(v1):
            a. (v1[i]-v2[I])^2 <--dist
    3)  Return dist^0.5
```

For function manhattan_distance

```
Function manhattan_distance(v1,v2):
    1) dist<--0
    2) manhattan_dist<--0
    3) For each I inlength(v1):
            a. Cal abs difference v2[i]-v1[I]
            b. Add abs difference to manhattan_dist
    4) Return manhattan_dist
```

For function label_encoding

```
Function label_encoding(cats):

    1)  labels<-- empty dict
    2)  enc_data<--empty list
    3) For each cat in cats:
            a. If cat is not in labels:
                I. Assign label length(labels)
                II. Add cat,labels to enc_data
    4) Return enc_data,labels
```

For function knn_classifier

```
Function knn_classifier(train_data,train_labels,test_inst,k):

    1) distances<--empty list
    2) For each I in length(train_data):
            a.  Calc euclidean dist train_data[i]-test_inst
            b.  Append (train_labels[i],dist) to distances
    3) neighbors<--empty list
    4) Repeat k times:
            a. min_dist<--infinity
            b. nearest_neighbor<--None
            c. For each label,dist in distances:
                I. If dist<min_dist and label not in neighbors:
                    i. Update min_dist to dist
                    ii. Update nearest_neighbor to (label,dist)
            d. Append nearest_neighbor to neighbors
    5) label_counts<-- empty dict
    6) For each label,_ in neighbors:
            a. If count>max_count:
                I. Update max_count to count
                II. Update most_common_label to label
    7) Return most_common_label
```

Function for one_hot_encoding

```
Function one_hot_encoding(labels):

    1) unique_labels<-- empty list
    2) For each label on labels:
            a. If label not in unique_labels:
                I. Append label to unique_labels
    3) encoding<-- empty list
    4) For each label in labels:
            a. one_hot<--empty list
            b. For each unique_label in unique_labels:
                I. If label==unique_label:
                    i. append 1 to one_hot
                II. Else:
                    i. Append 0 to one_hot
            c. Append one_hot to encoding
    5) Return encoding
```

User input prompt

For euclidean and manhattan distances

```
1) input for v1
2) Input for v2
3) Call the function euclidean_distance, calculate and display the result
4) Call the function manhattan_distance, calculate and display the result
```

For knn-classifier

```
1) Input for k
2) Input for cats
3) train_data,train_labels<--empty list
4) For each cat in cats:
    a. Input for train data for cat
    b. Input for label train data point
    c. Append train data,label to train_data,train_labels
5) Input for test instance
6) Call knn_classifier to predict label and display prediction
```

For label encoding

```
1) Input of cats
2) Encode cats calling label_encoding and display encoded data
```

For one hot encoding

```
1) Input of labels
2) Encode labels calling one_hot_encoding and display encoded labels
```