**22AIE213 – MACHINE LEARNING**

Lab Assignment Report – 02

**Submitted by**

Nossam Sri Chakradhar - BL.EN. U4AIE22132

Gopa Pulastya - BL.EN. U4AIE22116

Rishi Anirudh K - BL.EN. U4AIE22139

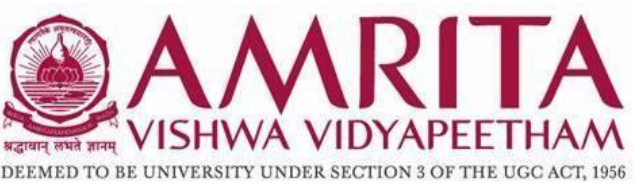
**For the fulfilment of the Course 22AIE213**

**Of**

**BACHELOR OF TECHNOLOGY**

**IN**

**“ARTIFICIAL INTELLIGENCE”**



AMRITA SCHOOL OF COMPUTING, BANGLORE

AMRITA VISHWA VIDYAPEETHAM

BENGALURU – 560035

**QUESTION-1:**

**Pseudocode:**

1. FUNCTION euclidian (a1, a2)

IF length(a1) != length(a2)

RETURN “Vectors must have same dimensionality.”

ELSE

FOR x, y IN zip(a1, a2)

sumofSquares + = (x - y)\*\*2

RETURN sqrt(sumofsquares)

END

1. FUNCTION manhattan (a1, a2)

IF length(a1) != length (a2)

RETURN “Vectors must have same dimensionality.”

ELSE

FOR x, y in zip(a1, a2):

sumofabsdifferences += abs(x - y)

RETURN sumofdifferences

END

1. FUNCTION getvector()

INPUT Vector

1. Create Instance of FUNCTION getvector()

INPUT Vector-1

INPUT Vector-2

1. RETURN Euclidian Distance FROM FUNCTION euclidian (Vector-1, Vector-2)
2. RETURN Manhattans Distance FROM FUNCTION Manhattan (Vector-1, Vector-2)
3. END

**Explanation:** This program finds the Euclidian and Manhattan Distance for the vectors that are inputted using the function Euclidian and Manhattan. First before we find the distance between the two distances first, we check the dimensions of both the vectors if the vectors are same, we find the distances but if the dimensions are not same, we print that the dimensionality should be same. The Euclidean distance is computed by summing the squares of the differences between corresponding elements and taking the square root of the sum. The Manhattan distance is calculated by summing the absolute differences between corresponding elements.

**QUESTION-2:**

**Pseudocode:**

1. FUNCTION euclidean(a1, a2)

distance = 0

FOR i FROM 0 TO length(a1) - 1

distance += (a1[i] - a2[i]) \*\* 2

RETURN distance \*\* 0.5

1. FUNCTION knn (train\_data, train\_labels, test\_point, k):

distances = []

FOR each data\_point, label in train\_data, train\_labels:

distance = euclidean\_distance(data\_point, test\_point)

distances.append((distance, label))

distances.sort() # Sort distances in ascending order

nearest\_neighbors = distances[:k] # Select k nearest neighbors

label\_counts = {}

FOR each neighbor in nearest\_neighbors:

label = neighbor[1] # Extract label from neighbor tuple

IF label in label\_counts:

label\_counts[label] += 1

ELSE:

label\_counts[label] = 1

predicted\_label = argmax(label\_counts) # Choose label with highest count

RETURN predicted\_label

1. FUNCTION getinput()

INPUT train\_data

INPUT train\_labels

INPUT test\_labels

INPUT k

FOR \_ IN range(num\_test):

features = list (map(float, input().split()))

X\_test.append(features)

k = int(input("Enter the value of k: "))

return X\_train, y\_train, X\_test, k

1. PREDICT the OUTPUT
2. END

**Explanation:** This program implements a k-nearest neighbors (KNN) classifier for classification tasks. The euclidean function computes the Euclidean distance between two points based on their dimensions, while the knnclassifier function utilizes this distance metric to classify test points by finding their nearest neighbors among the training data. User input is handled by the getinput function, allowing users to specify the number of data points, features, and the value of k. Finally, the main function orchestrates the entire process, prompting users for input, performing classification, and displaying the predicted labels for the test data.

**QUESTION-3**

**Pseudocode:**

1. FUNCTION labelencoding(data):

SET labelmap = {}

SET encodeddata = []

SET nextlabel = 0

FOR each row in data:

SET encodedrow = []

FOR each value in row:

IF value is not in labelmap:

Assign labelmap[value] = nextlabel

Increment nextlabel by 1

Append labelmap[value] to encodedrow

Append encodedrow to encodeddata

RETURN encodeddata, labelmap

1. FUNCTION getinput():

samples = INPUT samples

features = INPUT features

INTIALIZE data = []

FOR each sample from 1 to samples:

INITIALIZE sample

FOR each feature from 1 to features:

INPUT features

Append feature to sample

Append sample to data

RETURN data

1. FUNCTION main():

INIITALIZE getinput()

FOR each row IN encodeddata:

PRINT(row)

FOR each label, code IN labelmap:

PRINT(label + ": " + code)

1. END

**Explanation:** This program converts categorical variables to numeric using label encoding. The label encoding function iterates through the input data, assigning a unique label to each unique category and replacing the original values with their corresponding labels. The get input function collects user input for categorical data. Finally, the main function orchestrates the process by obtaining input data, performing label encoding, and displaying the encoded data along with the label map, which contains the mapping between original categorical values and their corresponding numeric labels.

**QUESTION-4:**

**Pseudocode:**

1. FUNCTION labelencoding (data)

INITIALIZE labelmap

INITIALIZE encodeddata

SET nextlabel TO 0

FOR each row IN data

INITIALIZE encodedrow

FOR each value IN row

IF value is not in labelmap:

SET labelmap[value] = nextlabel

Increment nextlabel by 1

RETURN encodeddata, labelmap

1. FUNCTION getinput():

INPUT samples

INPUT features

INITIALIZE data

PRINT ("Enter the complete categorical data:")

FOR each sample FROM 1 TO samples

INITIALIZE sample LIST

FOR each feature from 1 to features:

INPUT features

SET feature to sample

SET sample to data

RETURN data

1. FUNCTION main():

CREATE instance for getinput()

encodeddata, labelmap = labelencoding(data)

FOR each row in encodeddata:

PRINT(row)

FOR each label, code in labelmap:

PRINT(label code)

1. END

**Explanation:** This program finds label encoding to convert categorical data into numerical form efficiently. The labelencoding function iterates through the dataset, assigning a unique numerical label to each distinct category encountered, facilitating the conversion process. With user input handled by the getinput function, users can specify the dataset's size and features, ensuring adaptability to different data structures. The main function serves as the script's core, orchestrating data input, label encoding, and output display, providing a seamless workflow for categorical data transformation. Overall, this script offers a concise and practical solution for preprocessing categorical data in preparation for machine learning tasks.