

Machine Learning

Assessment-1



Date:08/02/24

**Team Members**

1. Vaishnavi- BL.EN.U4AIE22131
2. Gayatri- BL.EN.U4AIE22125
3. Hari Sankar- BL.EN.U4AIE22109

**CODE:**

*import* math

def euclidean\_distance(vector1, vector2):

*if* len(vector1) != len(vector2):

*raise* ValueError("Vectors must have the same dimension")

    distance = 0

*for* i *in* range(len(vector1)):

        distance += (vector1[i] - vector2[i])\*\*2

*return* math.sqrt(distance)

def manhattan\_distance(vector1, vector2):

*if* len(vector1) != len(vector2):

*raise* ValueError("Vectors must have the same dimension")

    distance = 0

*for* i *in* range(len(vector1)):

        distance += abs(vector1[i] - vector2[i])

*return* distance

def k\_nearest\_neighbors(train\_data, test\_instance, k):

    distances = []

*for* train\_instance, label *in* train\_data:

        distance = euclidean\_distance(train\_instance, test\_instance)

        distances.append((distance, label))

    distances.sort(key=lambda x: x[0])

    neighbors = distances[:k]

    class\_votes = {}

*for* \_, label *in* neighbors:

        class\_votes[label] = class\_votes.get(label, 0) + 1

*return* max(class\_votes, key=class\_votes.get)

def label\_encoding(categories):

    label\_map = {}

*for* i, category *in* enumerate(categories):

        label\_map[category] = i

*return* label\_map

def one\_hot\_encoding(categories):

    unique\_categories = list(set(categories))

    encoded\_vectors = []

*for* category *in* categories:

        encoded\_vector = [0] \* len(unique\_categories)

        index = unique\_categories.index(category)

        encoded\_vector[index] = 1

        encoded\_vectors.append(encoded\_vector)

*return* encoded\_vectors

*# main class*

*if* \_\_name\_\_ == "\_\_main\_\_":

*# Euclidean distance*

    vector1 = [1, 2, 3]

    vector2 = [4, 5, 6]

    print("Euclidean distance:", euclidean\_distance(vector1, vector2))

*# Manhattan distance*

    print("Manhattan distance:", manhattan\_distance(vector1, vector2))

*# k-NN classifier*

    train\_data = [([1, 2], 'A'), ([2, 3], 'B'), ([3, 4], 'A')]

    test\_instance = [1.5, 2.5]

    k = 2

    print("Predicted class label:", k\_nearest\_neighbors(train\_data, test\_instance, k))

*# Label encoding*

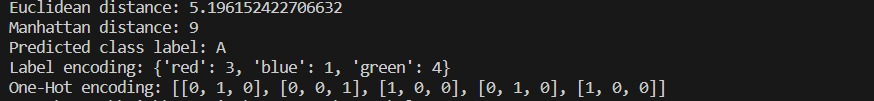
    categories = ['red', 'blue', 'green', 'red', 'green']

    print("Label encoding:", label\_encoding(categories))

*# One-Hot encoding*

    print("One-Hot encoding:", one\_hot\_encoding(categories))

**OUTPUT:**

****

**PSEUDO CODE:**

function euclidean\_distance(vector1, vector2):

if length(vector1)! = length(vector2):

raise ValueError ("Vectors must have the same dimension"

distance = 0

for i from 0 to length(vector1) - 1:

distance += (vector1[i] - vector2[i])^2

return sqrt(distance)

function manhattan\_distance(vector1, vector2):

if length(vector1)! = length(vector2):

raise ValueError ("Vectors must have the same dimension")

distance = 0

for i from 0 to length(vector1) - 1:

distance += abs(vector1[i] - vector2[i])

return distance

function k\_nearest\_neighbors (train\_data, test\_instance, k):

distances = []

for train\_instance, label in train\_data:

distance = euclidean\_distance (train\_instance, test\_instance)

distances.append ((distance, label))

distances.sort (key=lambda x: x[0])

neighbors = distances[:k]

class\_votes = {}

for label in neighbors:

class\_votes[label] = class\_votes.get (label, 0) + 1

return argmax(class\_votes)

function label\_encoding (categories):

label\_map = {}

for i from 0 to length(categories) - 1:

label\_map [categories[i]] = i

return label\_map

function one\_hot\_encoding (categories):

unique\_categories = unique(categories)

encoded\_vectors = []

for category in categories:

encoded\_vector = [0] \* length (unique\_categories)

index = index\_of (unique\_categories, category)

encoded\_vector [index] = 1

encoded\_vectors.append (encoded\_vector)

return encoded\_vectors

# main class

if \_\_name\_\_ == "\_\_main\_\_":

vector1 = [1, 2, 3]

vector2 = [4, 5, 6]

print("Euclidean distance:", euclidean\_distance(vector1, vector2))

print("Manhattan distance:", manhattan\_distance(vector1, vector2))

train\_data = [([1, 2], 'A'), ([2, 3], 'B'), ([3, 4], 'A')]

test\_instance = [1.5, 2.5]

k = 2

print("Predicted class label:", k\_nearest\_neighbors(train\_data, test\_instance, k))

categories = ['red', 'blue', 'green', 'red', 'green']

print("Label encoding:", label\_encoding(categories))

print("One-Hot encoding:", one\_hot\_encoding(categories))

**EXPLANATION:**

1. Euclidean and Manhattan Distance Functions:

Euclidean and Manhattan distance functions calculate the distance between two vectors. The Euclidean distance is calculated by adding the squared differences of the corresponding elements and multiplying them by the square root. The Manhattan distance, also known as the L1 norm, is calculated by multiplying the absolute difference of the corresponding elements by the square root of the squared differences.

2. K-Nearest Neighbor Classifier (k-NN):

K\_NEAREST\_NEIGHBORS (test\_data,test\_instance,k): K-NN classifier: k-nearest neighbors: Euclidean distance from a test instance to each training instance. Sort the distances, select k nearest neighbors, and return the class label with a majority vote among k nearest neighbors.

3. labels and one-hot-encoding functions:

Converts a set of categorical variables, such as colors, to a set of numerical labels using the label encoding function. Each category has its own unique numerical label. one-hot encoding (categories): In one-hot encoding, each category is converted to a set of numeric labels using the one-hot encoding function. The category is represented by a binary vector with only one element that indicates the presence of the category.

4. Main Section:

This section will show how to use the functions with the sample data. We will also show how to calculate the Euclidean distance and the Manhattan distance between two vectors. We will also use the k-NN to predict the class labels for a test instance based on a small training data set. Finally, we will show the label encoding for categorical variables and how one-hot encoding works for categorical variables.