Machine Learning Report

Assignment-2

Name : S.Hemanth Sai

Roll No. : AIE22171

1. **Pseudo Code**:

*function Euclidean\_Distance(V1, V2):*

*if length(V1) is not equal to length(V2):*

*print "Both vectors are not of same length"*

*distance = 0*

*for i from 0 to length(V1) - 1:*

*distance += (V1[i] - V2[i]) squared*

*return square root of distance*

**Euclidean\_Distance(V1, V2):**  
- Estimates Euclidean distance between two vectors.  
- Checks if the lengths are equal for both vectors.  
- It gives square root sums squared differences between the elements in vectors.

*function Manhattan\_Distance(V1, V2):*

*if length(V1) is not equal to length(V2):*

*print "Both vectors are not of same length"*

*distance = 0*

*for i from 0 to length(V1) - 1:*

*distance += absolute value of (V1[i] - V2[i])*

*return distance*

**Manhattan\_Distance(V1, V2):**  
- Computes the Manhattan distance between two vectors.  
- Determines whether vectors have the same length.  
- Sum of absolute values difference between elements included in two vectors.

1. **Pseudo Code**:

k-NN Classifier:

-

1. **Pseudo Code**:

*function Label\_Encoding(data):*

*unique\_values = empty list*

*for each val in data:*

*if val is not in unique\_values:*

*append val to unique\_values*

*print unique\_values*

*encoding = empty dictionary*

*index = 0*

*for each val in unique\_values:*

*encoding[val] = index*

*increment index by 1*

*encoded\_data = empty list*

*for each val in data:*

*append encoding[val] to encoded\_data*

*return encoded\_data*

**Label\_Encoding(data):**  
- One types it to numerical labels categorical variables in the input data.  
- Mixes unique values in the information and associates each exceptional value a component code that is unequivocally numbered.  
- It supplies a list number such as the label for each of categorical values, output is received.

1. **Pseudo Code**:

*function One\_Hot\_Encoding (data):*

*unique\_values = empty list*

*for each val in data:*

*if val is not in unique\_values:*

*append val to unique\_values*

*print unique\_values*

*encoding = empty dictionary*

*index = 0*

*for each val in unique\_values:*

*encoding[val] = list of length of unique\_values filled with zeros*

*encoding[val][index] = 1*

*increment index by 1*

*encoded\_data = empty list*

*for each val in data:*

*append encoding[val] to encoded\_data*

*return encoded\_data*

**One-Hot Encoding (data):**- One-hot encodes categorical variables in the data.  
- Encode each individual value with a binary code, place 1 where it is none zero and leave others at zeros.  
- It converts categorical values into one-hot encoded representation as an inner list.

Name:- CH SUBRAHMANYA SRESTI

Roll No:- AIE22179

Semester:-3

1.Psuedo Code:-

function EuclideanDistance(V1, V2):

sum\_of\_squares = 0

for i from 0 to length(V1) - 1:

diff = V1[i] - V2[i]

sum\_of\_squares += diff \* diff

euclidean\_distance = square\_root(sum\_of\_squares)

return euclidean\_distance

function ManhattanDistance(V1, V2):

sum\_of\_absolute\_diff = 0

for i from 0 to length(V1) - 1:

absolute\_diff = absolute\_value(V1[i] - V2[i])

sum\_of\_absolute\_diff += absolute\_diff

return sum\_of\_absolute\_diff

V1 = [1, 5, 3]

V2 = [3, 4, 5]

print("Euclidean Distance:", EuclideanDistance(V1, V2))

print("Manhattan Distance:", ManhattanDistance(V1, V2))

Explaination:

Euclidian\_distance: It iterates with every corresponding element of vector, sums the diff and

Calculates the square root of sum

Manhattan\_distance: It iterates with every element of vector and calculates the absolute

Difference and sums up the difference.

2.Psuedo Code:-

import numpy as np

from collections import Counter

class KNN\_Classifier:

    def \_\_init\_\_(self,k):

        self.k=k

    def fit(self, X\_train, y\_train):

        self.X\_train=X\_train

        self.y\_train=y\_train

    def predict(self, X\_test):

        y\_pred=[self.\_predict(x) for x in X\_test]

        return np.array(y\_pred)

    def \_predict(self,x):

        distances=[np.linalg.norm(x-x\_train) for x\_train in self.X\_train]

        k\_indices=np.argsort(distances[:self.k])

        k\_nearest\_labels=[self.y\_train[i] for i in k\_indices]

        most\_common=Counter(k\_nearest\_labels).most\_common(1)

        return most\_common[0][0]

X\_train=np.array([[1,2],[2,3],[3,4],[4,5]])

y\_train=np.array([0,0,1,1])

X\_test=np.array([[2.5,3.5],[1.5,2.5]])

knn=KNN\_Classifier(k=3)

knn.fit(X\_train,y\_train)

predictions=knn.predict(X\_test)

print("predictions:",predictions)

Explaination:-

A KNN classifier class that predicts labels for given test data based on the k-nearest neighbors algorithm. It calculates Euclidean distances between test points and training data, selects the k nearest neighbors, and returns the most common label among them. Finally, it demonstrates the classifier's functionality by training on provided data `X\_train` and `y\_train`, and making predictions for `X\_test`, printing the predictions.

3.Psuedo Code:-

Function Label\_Encoding(Category):

Initialize an empty dictionary Unique\_Values to store unique categories and their corresponding numerical labels.

Initialize an empty list Encode\_Data to store the encoded values.

For each category i in the input list Category:

If the category i is not already in Unique\_Values:

Assign a numerical label to the category i by assigning its length as its label (since it's the next available label).

Add the category i and its numerical label to Unique\_Values.

Append the numerical label corresponding to the category i to Encode\_Data.

Return Encode\_Data containing the encoded values.

Set Category\_Data to ["Vivek", "Sresti", "Sharma", "Sresti", "Vivek"].

Print "Label Encoded Values:" followed by the result of calling Label\_Encoding(Category\_Data).

Explaination:-

* Initialize an dictionary with Unique\_Values to store keys as Category and labels as values. Then it iterates i in input list Category and initialize Encode\_Data to store Encode Values. Add the category and its numerical label to **Unique\_Values**.
* Append the numerical label corresponding to the category to **Encode\_Data**

4.Psuedo Code:-

Function euclidean\_distance(v1, v2):

If lengths of vectors v1 and v2 are not equal, return infinity to indicate incomparability

Compute the Euclidean distance between v1 and v2

Return the computed distance

Function get\_neighbors(training\_data, test\_instance):

Initialize an empty list distances

For each data point in training\_data:

Compute the Euclidean distance between the data point and the test instance, and append it to distances

Return distances

Function Knn\_classifier(training\_data, test\_instance, k\_value):

Get the nearest neighbors of the test instance using get\_neighbors function

Sort the neighbors by distance

Select the k\_value nearest neighbors

Count the occurrences of each class among the nearest neighbors

Return the most common class among the nearest neighbors

Explaination:-

One Hot encoding does encoding the categorical values into numerical values that is it converts 1 where it has non zero value and places 0 which has zero value. It converts one hot encoding into inner list.

Name : V .Bhavaharnath Sharma

Roll No : AIE22162

1. **Pseudo Code**:

Function euclidean\_distance(v1, v2):

if length of v1 is not equal to length of v2:

raise ValueError("Vectors should be of same dimension")

sum\_of\_squares = 0

for each element x, y in zip(v1, v2):

sum\_of\_squares += (x - y) \*\* 2

euclidean\_distance = square\_root(sum\_of\_squares)

return euclidean\_distance

Function manhattan\_distance(v1, v2):

if length of v1 is not equal to length of v2:

raise ValueError("Vectors should be of same dimension")

total\_distance = 0

for each element x, y in zip(v1, v2):

total\_distance += absolute\_value(x - y)

return total\_distance

Function get\_vector\_from\_input():

prompt user to enter the vector (separated by commas)

read the input as vector\_str

split the vector\_str by comma and store it as a list of strings called vector\_str\_list

initialize an empty list called vector

for each string x in vector\_str\_list:

convert x to float, strip whitespace, and append it to vector

return vector

Print "Enter coordinates for vector 1:"

v1 = call get\_vector\_from\_input()

Print "Enter coordinates for vector 2:"

v2 = call get\_vector\_from\_input()

euclidean = call euclidean\_distance(v1, v2)

manhattan = call manhattan\_distance(v1, v2)

Print "Euclidean distance:" followed by euclidean

Print "Manhattan distance:" followed by Manhattan

**Explaination :**

This Python code presents functions to calculate Euclidean and Manhattan distances between two vectors v1 and v2. It prompts the user to input the coordinates of two vectors, calculates the distance and finally, the results are printed. The `get\_vector\_from\_input()` function gets user input for vector coordinates The `euclidean\_distance()` function calculates the Euclidean distance between two vectors, and the `manhattan\_distance()` function computes the Manhattan distance. If the dimensions of the input vectors are not matched a `ValueError` is raised.

1. **Pseudo Code**:

function euclidean\_distance(v1, v2):

if len(v1) != len(v2):

return infinity # Return a very large value to indicate incomparability

sum\_of\_squares = 0

for i in range(len(v1)):

sum\_of\_squares += (v1[i] - v2[i])^2

return sqrt(sum\_of\_squares)

function get\_neighbors(training\_data, test\_instance):

distances = []

for train\_data in training\_data:

distance = euclidean\_distance(train\_data[0], test\_instance)

distances.append((distance, train\_data[1])) # Appending distance and corresponding class label

distances.sort(key=lambda x: (x[0], x[1]))

return distances

function Knn\_classifier(training\_data, test\_instance, k\_value):

neighbors = get\_neighbors(training\_data, test\_instance)

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

classes = [neighbor[1] for neighbor in nearest\_neighbors] # Extract class labels of nearest neighbors

class\_counter = Counter(classes)

most\_common\_class = class\_counter.most\_common(1)[0][0] # Get the most common class label among nearest neighbors

return most\_common\_class

training\_data = [([140, 60], 'medium'), ([145, 55], 'medium'), ([165, 70], 'large'), ([170, 65], 'large'), ([155, 72], 'large')]

test\_instance = [160, 58]

k\_value = 2

result = Knn\_classifier(training\_data, test\_instance, k\_value)

print("Predicted class:", result)

-

**Explaination :**

In the above python code , The `euclidean\_distance` function computes the Euclidean distance between two vectors v1 and v2. The `get\_neighbors` function calculates distances between the test instance and all training instances, sorts them by distance and class label and returns the sorted list. `Knn\_classifier` function uses `get\_neighbors` operation to select `k\_value` closest neighbors.The KNN classifier predicts the class label of `test\_instance` by looking at the majority class among `k\_value` nearest neighbors in `training\_data` which is then printed.

1. **Pseudo Code**:

*FUNCTION label\_encode\_categorical(data):*

*labels = {}*

*label\_count = 0*

*encoded\_data = []*

*FOR EACH value IN data:*

*IF value NOT IN labels:*

*labels[value] = label\_count*

*label\_count += 1*

*encoded\_data.append(labels[value])*

*RETURN encoded\_data*

*# Example*

*categorical\_data = ['India', 'Australia', 'England', 'India', 'England', 'Australia']*

*numeric\_data = label\_encode\_categorical(categorical\_data)*

*PRINT "Original categorical data:", categorical\_data*

*PRINT "Numeric data after label encoding:", numeric\_data*

**Explaination :**

In the above python code , The ` label\_encode categorical` function iterates through each value in the input data. It assigns an unique numeric label to each unique categorical literal of the input, using an incremented counter. The function then returns a list where each list element is the assigned numeric label corresponding to the respective categorical value in the input data. It prints the original categorical data and also the corresponding numeric labels obtained by label encoding.

1. **Pseudo Code**:

*function one\_hot\_encoding(data):*

*unique\_values = list(set(data))*

*encoded\_data = []*

*for value in data:*

*encoding = [0] \* len(unique\_values)*

*index = unique\_values.index(value)*

*encoding[index] = 1*

*encoded\_data.append(encoding)*

*return encoded\_data*

*categorical\_data = ['WI', 'IND', 'SL', 'WI', 'ENG', 'IND']*

*encoded\_data = one\_hot\_encoding(categorical\_data)*

*for i, encoded\_value in enumerate(encoded\_data):*

*print(f'{categorical\_data[i]}: {encoded\_value}')*

**Explaination :**

In the above python code , It first identifies unique values in input data. After that, it runs through each value in the data and generates a one-hot encoded vector matching it where the index corresponding to the value's position in the unique values list is set to 1, and all other indices remain unchanged. These encodings are then appended to the list. It returns a list of encoded data in the form one-hot encoding of the corresponding input value.