MACHINE LEARNING

22AIE213

Assignment 2

Team-16

[Thanuj R – 22140, Prem Reddy YS – 22166, Mani Sankar D - 22112]

1.Write a function to calculate the Euclidean distance and Manhattan distance between two vectors. The vectors dimension is variable. Please don’t use any distance calculation functions available in Python.

Algorithm for Euclidean\_Distance:

1. Initialize variable c to 0

2. For each index i from 0 to the length of vectors a and b - 1:

a. Increment c by the square of the difference between a[i] and b[i]

3. Return the square root of c using the math.sqrt function

Algorithm for Manhattan\_Distance

1. Initialize variable distance to 0

2. For each index i from 0 to the length of vectors a and b - 1:

a. Increment distance by the absolute difference between a[i] and b[i]

3. Return distance

Main input part:

1. Input the dimensions of the vectors n

2. Initialize vectors a and b with n dimensions

3. For each index i from 0 to n - 1:

a. Input the element a[i]

b. Input the element b[i]

4. Call Euclidean\_Distance function with vectors a and b and store the result in ans

5. Call Manhattan\_Distance function with vectors a and b and store the result in ans1

6. Print "Euclidean distance"

7. Print "Manhattan distance "

**Explanation:**

In this algorithm we have written the code for the Euclidean and the manhattan distance using the simple basic math functions like “sqrt” and “abs” etc. Using these simple functions we return the distances to the main function and to display the output of the code.

2.Write a function to implement k-NN classifier. k is a variable and based on that the count of neighbors should be selected.

Algorithm for Euclidean Distance:

1. Initialize variable c to 0

2. For each index i from 0 to the length of vectors a and b - 1:

a. Increment c by the square of the difference between a[i] and b[i]

3. Return the square root of c using the math.sqrt function

Algorithmfor k\_nearest\_neighbors

1. Initialize an empty list distances

2. For each instance i in Numbers\_train:

a. Calculate the Euclidean distance between x\_test and Numbers\_train[i] using Euclidean\_Distance function

b. Append a tuple (distance, y\_train[i]) to the distances list

3. Sort distances in ascending order based on the distance values

4. Select the first k neighbors from distances

5. Print the first k neighboring distances in the given instances

6. Initialize an empty dictionary class\_counts

7. For each neighbor in neighbors:

a. Get the label from the neighbor

b. Increment the count of the label in class\_counts

8. Return the label with the highest count as the predicted label

Main Input:

1. Initialize training instances Numbers\_train as np.array([[1, 2], [2, 3], [3, 4], [4, 5]])

2. Initialize corresponding labels y\_train as np.array(['A', 'A', 'B', 'B'])

3. Initialize test instance test as np.array([2.5, 3.5])

4. Initialize k\_value as 3

5. Call k\_nearest\_neighbors function with Numbers\_train, y\_train, test, and k\_value and store the result in predicted\_class

6. Print "The predicted class for the test point is:"

**Explanation:**

At first we take in the training features as arrays in the input part and Also the K value which decides the number of values to be considered for the prediction. Then first we find the Eucledian Distances for all the instances in the given training set. Then according to the number value of k we take the 1st k values which we have put it in ascending order and using those values , using the Discrete value function formula to find the prediction.

3.Write a function to convert categorical variables to numeric using label encoding. Don’t use any existing functionalities.

Algorithm for label Encoding :

1. Initialize an empty list b with the same length as list a

a. b = [0 for i in range of length a]

2. Initialize an empty list Unique\_set

3. For each element i in list a:

a. If i is not in Unique\_set:

i. Append i to Unique\_set

4. For each index i from 0 to the length of list a - 1:

a. For each element j in Unique\_set:

i. If a[i] is equal to j:

- Set b[i] to the index of j in Unique\_set

5. Return list b

Main Part For initializing the code:

1. Initialize list a with values ["red", "blue", "red", "green", "blue", "green", "red"]

2. Call label\_Encoding function with list a and store the result in list b 3. Print "The result using label encoding:"

**Explanation:**

Using the Label encoding we can easily convert the labels to the Numeric and easily use it while doing any manipulations using the given data. We first find the Unique Elements from the set and set it in another set. Then using that set we give the numbering to the elements.

**4.Write a function to convert categorical variables to numeric using One-Hotencoding. Don’t use any existing functionalities**.

Algorithm for one hot encoding:

1. Initialize an empty list b

2. Initialize an empty list Unique\_set

3. For each element i in list a:

a. If i is not in Unique\_set:

i. Append i to Unique\_set

4. For each index i from 0 to the length of list a - 1:

a. Initialize a category list with all zeros of length len(Unique\_set)

b. For each element j in Unique\_set:

i. If a[i] is equal to j:

- Set the corresponding index in the category list to 1

c. Append the category list to list b

5. Return list b

Main Part For Input:

1. Initialize list a with values ["red", "blue", "red", "green", "blue", "green", "red"]

2. Call one\_hot\_encoding function with list a and store the result in list b

3. Print "The result using one-hot encoding:"

**Explanation:**

Using the one hot encoding we can easily convert the labels to the Numeric and easily use it while doing any manipulations using the given data. We first find the Unique Elements from the set and set it in another set. Then using that set we give the numbering to the elements. Here we give the index number of the category list as high or 1 and add it to the result list.