Machine Learning Assignment – 02

Team:

BL.EN.U4AIE22118 - Ishvarya G

BL.EN.U4AIE22121 - Kavya Sree Kammari

BL.EN.U4AIE22151 - Sharanya V Thambi

Kavya Sree Kammari

Question – 1:

The code defines the functions euclidean_distance, manhattan_distance. that calculates the Euclidean distance between two vectors. Euclidean distance, also known as L2 norm, is a measure of the straight-line distance between two points in n-dimensional space. The function checks if the dimensions of the input vectors are the same, and if not, it raises a ValueError. The Manhattan_distance computes the sum of squares of the differences between corresponding elements of the vectors, takes the square root of this sum, and returns the Euclidean distance. Calculates the sum of absolute differences between corresponding elements of the vectors.

- Returns the sum of absolute differences, which represents the Manhattan distance.

Pseudocode:

```
Function euclidean_distance(vector1, vector2):
    Find the length of the vectors
    if length_1 ≠ length_2:
        Raise ValueError
    sum_of_squares = 0
    for i = 0 to length_1 - 1:
        sum_of_squares += (vector1[i] - vector2[i])^2

    euclidean_dist = square root of sum_of_squares
    return euclidean_dist


function manhattan_distance(vector1, vector2):
   Find the length of the vectorsd
    if length_1 ≠ length_2:
        Raise ValueError
    sum_of_differences = 0
    for i = 0 to length_1 - 1:
        sum_of_differences += absolute value of (vector1[i] - vector2[i])

    return sum_of_differences
```

Question – 2:

In the code function named knn_classify, which implements the k-nearest neighbors (KNN) classification algorithm. This function takes three parameters: the training data set (training_data), a single test instance (test_instance), and the value of k (k) representing the number of nearest neighbors to consider for classification. It computes the Euclidean distance between the test instance and each data point in the training set, sorts the distances in ascending order, selects the k nearest neighbors, and then tallies the votes for each class label among these neighbors. Finally, it returns the class label that received the highest number of votes as the predicted label for the test instance.

Pseudocode:

```
Procedure knn_classify(training_data, test_instance, k):
    distances = empty list  # List to save distance between each point
    for each row in training_data:
        Compute the Euclidean distance between the current training data point and the test instance
        Append a tuple containing the training data point and its distance to the distances list
    Sort the distances list based on the distance in ascending order
    Select the k nearest neighbors from the sorted distances list
    class_votes = empty dictionary  # Dictionary to store votes for each class label

    for each neighbor in the k nearest neighbors:
        Extract the class label of the neighbor
        If the label is already in class_votes:
            Increment its vote count
        Else:
            Initialize its vote count to 1

    Return the class label with the maximum number of votes
```

Question – 3:

In this code, The function encodes categorical variables in the input DataFrame using label encoding. It replaces each unique categorical value in the DataFrame columns with a corresponding integer label. It returns the encoded DataFrame along with a dictionary containing label encoders for each column.

Pseudocode:

```
Procedure label_encoding(data):
    encoded_data = make a copy of data  # Avoid modifying the original data directly
    label_encoders = empty dictionary

    for each column in encoded_data:
        if data type of column is 'object':  # Check if the column contains categorical values
            unique_labels = unique values in column
            label_encoder = create a label encoder mapping each label to a unique integer
            map labels in column to integer values using label_encoder
            store label_encoder in label_encoders dictionary
    return encoded_data, label_encoders
```

Question – 4:

This function performs one-hot encoding on categorical variables in the input DataFrame. It creates binary columns for each unique value in the categorical columns, indicating the presence or absence of that value in the original column. It returns the encoded DataFrame with one-hot encoded columns and a list of the names of these new columns.

Pseudocode:

```
Procedure one_hot_encoding(data):
    encoded_data = make a copy of data  # Avoid modifying the original data directly
    one_hot_encoded_columns = empty list

    for each column in encoded_data:
        if data type of column is 'object':  # Check if the column contains categorical values
            unique_values = unique values in column
            for each value in unique_values:
                new_column_name = column_name + '_' + value  # Create a new column name
                create a new binary column indicating presence of value
                append new_column_name to one_hot_encoded_columns
            drop the original column from encoded_data

    return encoded_data, one_hot_encoded_columns
```

Ishvarya G

In this program for questions 1,3 and 4, I have made the code modularized specific to each question's functionality. The main function provides the functionality of taking input from the user's end for each function and a call is made to the respective function to perform the operation, the result of which is displayed using the main function. However, for implementation of the K-NN classifier (Question 2) there is no main function. The code uses custom fit(), predict() functions without using any external ML libraries (except pandas that is used to load in the train and test csv files) for its implementation.

Question 1:

To calculate the Euclidean and Manhattan distance between 2 vectors whose dimensions are variable :

Explanation:

The code uses 2 For loops to loop through the range of numbers from 0 till the length of the vector. Using the formula of taking the sum of the squares of differences of elements of the 2 given vectors Euclidean distance is calculated. Similarly, for finding the Manhattan distance, the sum of the absolute differences of the vector elements is taken. Finally the calculated values are returned.

Pseudocode:

```
Function findEuclidean( INPUT vec1,vec2):
        SET euclidean=0

FOR i in the range of length of vec2:

#since the length of both the vectors must be equal : length of vec1 = length of vec2

SET euclidean = euclidean + (vec2[i] – vec1[i]) ** 2

REPEAT until i reaches the length of vec2

RETURN sqrt(euclidean)
```

```
Function findManhattan( INPUT vec1,vec2):
        SET manhattan=0

FOR j in the range of length of vec2:

#since the length of both the vectors must be equal : length of vec1 = length of vec2

SET manhattan= manhattan + abs(vec2[j] – vec1[j])

REPEAT until i reaches the length of vec2

RETURN sqrt(manhattan)
```

Question 2:

To implement K-NN classifier using a function :

Explanation:

The code defines a KNNclassifier class that initializes k (no.of neighbours to check for), fit() that just stores the train and test data to be used during predictions and the predict() function that has the function to calculate the Euclidean distances between each feature of the training and test samples and returns distances_for_test_point list, which is further sorted in ascending order. Then, we extract the indices of the first ‘k’ closest test points and also extract its corresponding labels. Among these, we find the majority of all the labels (taking the max()) and add it to the predictions_list for the test set.

Pseudocode :

```
class KNNclassifier:

        Function __init__(INPUT k):

                Initialize k


        Function fit(INPUT X_train, y_train):

                Store X_train and y_train as instance variables
```

```
    Function predict(INPUT X_test,X_train):
        Function findEuclidean(vec1, vec2):
            #same as in Question 1
        Convert train and test set to NumPy arrays
        SET predictions_list to []
        FOR test_point in test_features:
            SET distances_for_test_point to []
            FOR train_point in train_features:
                SET distance to the result of the Euclidean distance between train and test points
                Append to the distances_for_test_point list
                SET sorted_indices to indices sorted in test_features according to the distances
                SET k_nearest_indices to the first k closest indices
                SET unique_labels = set(k_nearest_neighbours)
                SET prediction = max(unique_labels, count of k_nearest_labels as the key)
                Append predictions to predictions_list
        RETURN predictions_list
Load the data
Create the KNN classifier object
Call fit() method on X_train and y_train
Call the predict() method on X_test and X_train
DISPLAY the returned predictions
DISPLAY the accuracy
```

Question 3:

To convert categorical variables to numeric using label encoding :

Explanation:

This function aims to encode the categorical inputs to numeric values by converting assigning a unique number to each unique label in the data. It first finds the unique_labels by using the set() method and uses For loop to iterate through the unique labels and sets the corresponding index as its value.

Pseudocode :

```
  Function encodeLabel (INPUT categ):
        SET label_mapping  = {}
SET unique_labels to the unique labels in categ using set()
FOR index ,label in enumerate(unique_labels):
SET label_mapping[label]  to index


SET encoded_labels to [label_mapping[label]  for label in categ]
        RETURN encoded_labels
```

Question 4:

To convert categorical variables to numeric using One-hot encoding :

Explanation:

This function aims to encode the categorical inputs to numeric values by converting assigning a binary value to each unique label in the data. It first finds the unique_labels by using the set() method then creates a list of lists that is filled with 0s whose row length is equal to the number of categorical variables and the column length is equal to the number of unique variables. We set the label_mapping to be equal to the corresponding index value. Then using 2 For loops we check if the value of j (iterable) matches the index of the label for current element in categ (categ[i]) , if so it sets the element in position i, j to 1. Finally it returns the encoded_labels.

Pseudocode:

```
Function encodeOneHot (INPUT categ):
    SET unique_labels to the unique labels in categ using set()
    SET encoded_labels to 0 where the rows is the length of items in categ and columns is the length of items in unique_labels
    SET label_mapping to {}


    FOR index ,label in enumerate(unique_labels):
SET label_mapping[label]  to index
    FOR i in range of length of categ:
            FOR j in range of length of unique_labels:
Check if j matches the index of the label for the current element in categ:
SET encoded_labels[i][j] = 1


RETURN encoded_labels
```

Sharanya VT

Question 1:

Pseudo code:

```
def euciladean(vector1,vector2,n):
sum initaialised to 0
        loop i from 0 to n:
            Assign sum as sum+power of 2 of corresponding values of vectors subtracted
        Return square root of sum

    def manhattan(vector1,vector2,n):
sum initaialised to 0
        loop i from 0 to n:
                Assign sum as sum+absolute value of corresponding values of vectors subtracted
        Return sum
```

Question 2:

Pseudo code:

```
Def knnclassifier_sharanya():
Insitialsie a map for label encoding of targer column ob BMI index
        Change index column to the numerical values that was encoded
        Create column for male and female and initialse it to 0
        One hot encode it according to the gender value for each instance
        Drop gender column
        Reorder the columns and bring male and female to the beginning
        Take a test vector to test the model
        Create a distance column and initialise it to 0
        Loop through the dataframe by calling indexes from 0 to length of dataframe:
            Call the eucliandean() and find ditance between testvector and
            instance_vector


        Sort the data frame according to distance


        Loop i through all values in map:
        Loop j through 0 to k:
if (i==Index value of that instance):
sum=sum+1
save sum in a a list sum_list


Max value of sum_list is found and then then position is stored


predicted target is the key corresponding to the position as value in map


Predicted target is returned
```

Question 3:

Pesudo code: