MACHINE LEARNING

22AIE213

Assignment -2- Report

Team Members -

Amara Gnana Sirishma – AIE22105

Yoshitha Tulasi- AIE22177

Meka Sai Sri Hanish- AIE22130

```
function euclidean_distance(vector1, vector2):
    squared_distances = [(a - b)^2 for a, b in zip(vector1, vector2)]
    return sqrt(sum(squared_distances))

function manhattan_distance(vector1, vector2):
    distance = sum(abs(x - y) for x, y in zip(vector1, vector2))
    return distance

function label_encoding(data, column):
    unique_labels = unique(data[column])
    encoding_dict = {}
    for i, label in enumerate(unique_labels):
        encoding_dict[label] = i
    data[column] = map(encoding_dict.get, data[column])
    return data

function one_hot_encoding(data, column):
    one_hot_encoded = create_one_hot(data[column])
    data = concatenate(data, one_hot_encoded, axis=1)
    data = drop_column(data, column)
    return data

function knn_classifier(X_train, y_train, X_test, k):
    distances = []
    for i in range(size(X_train, 0)):
        distance = euclidean_distance(X_train[i], X_test)
        distances.append((distance, y_train[i]))

    sort(distances, key=lambda x: x[0])
    neighbors = distances[:k]
```

```
    neighbor_labels = [neighbor[1] for neighbor in neighbors]

    prediction = mode(neighbor_labels)
    return prediction

if __name__ == "__main__":
    # Load dataset

    # Features (X) and Labels (y)

    # Convert categorical variables to numeric using label encoding

    # Convert categorical variables to numeric using One-Hot encoding

    # Split data into training and testing sets

    # Standardize features

    # Display original and encoded data

    # Calculate Euclidean distance for an example

    # Calculate Manhattan distance for an example

    # Example: Use k-NN classifier with k=3

    # Print the prediction
```

Explanation-

I have taken the data set to be the BMI weight details. It has got 4 columns, namely- gender, height, weight and BMI index. It contains 501 rows. So basically, this code implements a k-NN (k-Nearest Neighbors) classifier for predicting BMI index categories using a dataset. The code first loads the dataset and preprocesses categorical variables like gender using both label encoding and one-hot encoding. It then splits the data into training and testing sets, standardizes the features using StandardScaler, and displays the original, label-encoded, and one-hot encoded data for the last 10 rows. The code calculates Euclidean and Manhattan distances between the last two rows in the standardized test set and demonstrates the k-NN classifier prediction for the last row using a k-value of 3.

```
# Function to calculate Euclidean distance
function euclidean_distance(vector1, vector2):
    if length(vector1) ≠ length(vector2):
        raise ValueError("Vectors must have the same dimension")
    squared_distance = 0
    for each element (x, y) in zip(vector1, vector2):
        squared_distance += (x - y) ^ 2
    return square_root(squared_distance)
# Function to calculate Manhattan distance
function manhattan_distance(vector1, vector2):
    if length(vector1) ≠ length(vector2):
        raise ValueError("Vectors must have the same dimension")
    distance = 0
    for each element (x, y) in zip(vector1, vector2):
        distance += absolute_value(x - y)
    return distance
# Function to implement k-NN classifier
function knn_classifier(training_data, test_instance, k):
    distances = []
    for each (training_instance, label) in training_data:
        distance = euclidean_distance(test_instance, training_instance)
        distances.append((distance, label))
    sorted_distances = sort(distances, key=lambda x: x[0])
    k_nearest_labels = []
    for _, label in sorted_distances[:k]:
        k_nearest_labels.append(label)
    predicted_label = mode(k_nearest_labels)
    return predicted_label

# Function to convert categorical variables to numeric using label encoding
function label_encoding(categories):
```

```
 label_map = {}
    for index, category in enumerate(unique(categories)):
        label_map[category] = index
    encoded_values = []
    for category in categories:
        encoded_values.append(label_map[category])
    return encoded_values
# Function to convert categorical variables to numeric using One-Hot encoding
function one_hot_encoding(categories):
    unique_categories = unique(categories)
    encoded_matrix = []
    for category in categories:
        category_index = index_of(unique_categories, category)
        encoded_row = []
        for i in range(length(unique_categories)):
            encoded_row.append(if i == category_index then 1 else 0)
        append(encoded_matrix, encoded_row)
    return encoded_matrix
# Example usage in the main program
if __name__ == "__main__":
    # Example data for testing
    vector1 = [1, 2, 3]
    vector2 = [4, 5, 6]
    print("Euclidean Distance:", euclidean_distance(vector1, vector2))
    print("Manhattan Distance:", manhattan_distance(vector1, vector2))
    training_data = [([1, 2], 0), ([4, 5], 1), ([7, 8], 0), ([10, 11], 1)]
    test_instance = [3, 4]
    k_value = 3
    print("k-NN Classifier Prediction:", knn_classifier(training_data, test_instance, k_value))
    categorical_data = ["red", "green", "blue", "red", "green"]
    print("Label Encoded Data:", label_encoding(categorical_data))
```

```
one_hot_encoded_data = one_hot_encoding(categorical_data)

print("One-Hot Encoded Data:")

print_matrix(one_hot_encoded_data)
```

Explanation-

It helps you measure how different two sets of numbers are using Euclidean and Manhattan distances. An example at the end shows how you can use these tools with some pretend data, like comparing distances between two sets of numbers and predicting a category for a new set of numbers. The example at the end shows how you can use these tools with some pretend data, like comparing distances between two sets of numbers and predicting a category for a new set of numbers.