Machine learning LAB-07

Perumalla Dharan
AP21110010201

1. Implement K-NN Classifier for classification of any dataset of your choice.
    a. Load an existing data set
    b. Split the data set to train and test sets
    c. Test your model using test set. Find accuracy and confusion Matrix.
    d. Examine the effect of the value of K on accuracy/performance. Plot the curve "k vs accuracy" and find out the value of k for maximum accuracy for the test samples.

NOTE: Don't use any library. Develop a generalised function to implement K-NN Classifier.

```
import csv
import random
import math
import matplotlib.pyplot as plt

# Step 1: Load an existing dataset
def load_dataset(filename):
    dataset = []
    with open(filename, 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)  # Skip the header row
        for row in csv_reader:
            dataset.append(row)
    return dataset

# Step 2: Split the dataset into train and test sets
def train_test_split(dataset, split_ratio=0.8):
    train_size = int(len(dataset) * split_ratio)
    train_set = []
    test_set = list(dataset)
```

```
    while len(train_set) < train_size:
        index = random.randrange(len(test_set))
        train_set.append(test_set.pop(index))
    return train_set, test_set

# Step 3: Implement the K-NN algorithm
def euclidean_distance(instance1, instance2):
    distance = 0
    for i in range(len(instance1) - 1):
        if instance1[i].isdigit():  # Check if the value is
numeric
            distance += (float(instance1[i]) -
float(instance2[i])) ** 2
    return math.sqrt(distance)

def get_neighbors(train_set, test_instance, k):
    distances = []
    for train_instance in train_set:
        dist = euclidean_distance(test_instance,
train_instance)
        distances.append((train_instance, dist))
    distances.sort(key=lambda x: x[1])
    neighbors = []
    for i in range(k):
        neighbors.append(distances[i][0])
    return neighbors

def predict_classification(neighbors):
    class_votes = {}
    for neighbor in neighbors:
        class_label = neighbor[-1]
        if class_label in class_votes:
            class_votes[class_label] += 1
        else:
```

```
            class_votes[class_label] = 1
    sorted_votes = sorted(class_votes.items(), key=lambda
x: x[1], reverse=True)
    return sorted_votes[0][0]

# Step 4: Test the model and find accuracy and confusion
matrix
def get_predictions(train_set, test_set, k):
    predictions = []
    for test_instance in test_set:
        neighbors = get_neighbors(train_set, test_instance,
k)
        prediction = predict_classification(neighbors)
        predictions.append(prediction)
    return predictions

def get_accuracy(test_set, predictions):
    correct = 0
    for i in range(len(test_set)):
        if test_set[i][-1] == predictions[i]:
            correct += 1
    return (correct / float(len(test_set))) * 100.0

def confusion_matrix(test_set, predictions):
    actual_values = [instance[-1] for instance in test_set]
    unique_classes = list(set(actual_values))
    num_classes = len(unique_classes)
    matrix = [[0] * num_classes for _ in
range(num_classes)]
    for i in range(len(actual_values)):
        actual_class_index =
unique_classes.index(actual_values[i])
        predicted_class_index =
unique_classes.index(predictions[i])
```

```
        matrix[actual_class_index][predicted_class_index]
+= 1
    return matrix

# Step 5: Examine the effect of the value of K on
accuracy/performance
def k_vs_accuracy(train_set, test_set, max_k):
    accuracies = []
    for k in range(1, max_k + 1):
        predictions = get_predictions(train_set, test_set,
k)
        accuracy = get_accuracy(test_set, predictions)
        accuracies.append(accuracy)
    return accuracies

def plot_k_vs_accuracy(accuracies, max_k):
    k_values = list(range(1, max_k + 1))
    plt.plot(k_values, accuracies, marker='o')
    plt.title('K vs Accuracy')
    plt.xlabel('K')
    plt.ylabel('Accuracy')
    plt.xticks(range(1, max_k + 1))
    plt.grid(True)
    plt.show()

# Load the dataset
# dataset =
load_dataset('/e:/SRM/Machine_Learning/Lab/Lab-7/dataset.cs
v')
dataset=
load_dataset('E:\SRM\Machine_Learning\Lab\Lab-8\iris.csv')

# Split the dataset into train and test sets
```

```
train_set, test_set = train_test_split(dataset,
split_ratio=0.8)

# Define the maximum value of K to examine
max_k = 20

# Calculate accuracy for different values of K
accuracies = k_vs_accuracy(train_set, test_set, max_k)

# Plot K vs Accuracy
plot_k_vs_accuracy(accuracies, max_k)

# Find the value of K for maximum accuracy
optimal_k = accuracies.index(max(accuracies)) + 1
print(f"Optimal value of K for maximum accuracy:
{optimal_k}")
```

```
PS E:\SRM\Machine_Learning> python -u "e:\SRM
Optimal value of K for maximum accuracy: 3
PS E:\SRM\Machine_Learning>
```