**JINNAH UNIVERSITY FOR WOMEN**



**COURSE:** Artificial Intelligence

**PROJECT TITLE:** Fruit Grading System

* **Group Members**

**1**-Umm e Ruman Asif

**2**-Anzila Alvi

**3**-Tehreem Altaf

**4**-Areeba Khan

* **Group leader:**

1. Anzila Alvi

* **Division of work among group members**

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | TASK | NAME | DURATION |
| 1 | Overall project coordination and algorithm implementation | **Umm-e-ruman** | **4 days** |
| 2 | User interface and input handling | **Areeba khan** | **3 days** |
| 3 | KNN algorithm implementation and code documentation | **Tahrim Altaf** | **2 days** |
| 4 | Dataset creation and testing | **Anzila Alvi** | **2 days** |

* **CHAPTER 1:Introduction**
* **Project Objective**

The objective of the project is to implement a Text-Based KNN (K-Nearest Neighbors) Fruit Grading System. This system takes user input for a selected fruit, allows the user to input the sweetness level, and then uses a KNN algorithm to predict the sweetness grade of the fruit based on the sweetness levels of its nearest neighbors.

* **Scope**

The scope of the project includes creating a sample set of fruits with associated attributes such as weight and sweetness. The system allows users to input their preferred fruit and sweetness level and employs a KNN algorithm to predict the sweetness grade.

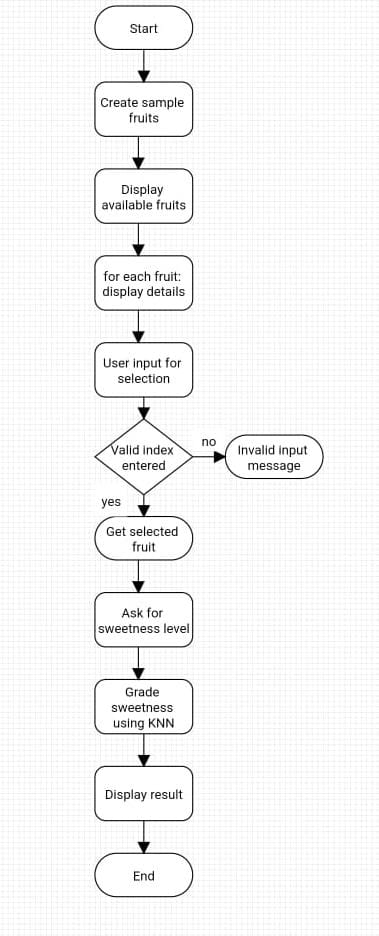
* **Target Audience**

The target audience for this project include individuals interested in exploring a simple text-based system for grading fruits based on sweetness levels.

* **Complete Description**

The Text-Based KNN Fruit Grading System provides a text interface for users to interact with the system. It displays a list of available fruits with their weights and prompts users to select a fruit by entering the corresponding index. After selecting a fruit, users are asked to input the sweetness level. The system then grades the sweetness of the selected fruit using a KNN algorithm, providing a grade (A, B, or C) based on the majority vote of the sweetness grades of its nearest neighbors.

* **CHAPTER 2:Flowchart**



* **CHAPTER 3:Implementation**
* **Dataset Description**

The dataset consists of a sample set of fruits with attributes such as name, weight, and initial sweetness level. This dataset is used for demonstration purposes and can be expanded with additional fruits and attributes if needed.

* **Methodology**

The system employs a KNN algorithm to predict the sweetness grade of a selected fruit. It calculates the distance between the sweetness levels of the selected fruit and all other fruits in the dataset. The K-nearest neighbors are then determined based on these distances. The sweetness grades of the nearest neighbors are used for a majority vote, and the predicted grade is displayed.

* **Algorithms**

**K-Nearest Neighbors (KNN):** Used for predicting sweetness grades based on the sweetness levels of nearest neighbors.

**Rule-Based Grading:** A rule-based grading logic is applied to individual sweetness levels to assign grades (A, B, or C).

* Code
* import java.util.ArrayList;
* import java.util.Collections;
* import java.util.List;
* import java.util.Scanner;
* public class FGS {
* public static void main(String[] args) {
* System.out.println("Text-Based KNN Fruit Grading System");
* // Sample fruits for demonstration
* List<Fruit> fruits = createSampleFruits();
* // Display available fruits
* System.out.println("Available Fruits:");
* for (int i = 0; i < fruits.size(); i++) {
* Fruit fruit = fruits.get(i);
* System.out.printf("%d. %s - Weight: %.2fg%n", i, fruit.getName(), fruit.getWeight());
* }
* // User input for fruit selection
* Scanner scanner = new Scanner(System.in);
* System.out.print("Enter the index of the fruit you want to view: ");
* if (scanner.hasNextInt()) {
* int selectedIndex = scanner.nextInt();
* if (selectedIndex >= 0 && selectedIndex < fruits.size()) {
* // Get the selected fruit
* Fruit selectedFruit = fruits.get(selectedIndex);
* // Ask for sweetness level
* System.out.print("Enter the sweetness level of the fruit: ");
* double sweetness = scanner.nextDouble();
* selectedFruit.setSweetness(sweetness);
* // Display the details of the selected fruit
* System.out.printf("%nDetails of %s:%n", selectedFruit.getName());
* System.out.printf("Weight: %.2fg%n", selectedFruit.getWeight());
* System.out.printf("Sweetness: %.2f%n", selectedFruit.getSweetness());
* // Grade the sweetness of the fruit using KNN
* char grade = gradeSweetnessKNN(selectedFruit, fruits);
* System.out.printf("Grade based on sweetness using KNN: %c%n", grade);
* } else {
* System.out.println("Invalid input. Please enter a valid index.");
* }
* } else {
* System.out.println("Invalid input. Please enter a valid index.");
* }
* }
* private static List<Fruit> createSampleFruits() {
* List<Fruit> fruits = new ArrayList<>();
* fruits.add(new Fruit("Apple", 150, 0.0));
* fruits.add(new Fruit("Banana", 120, 0.0));
* fruits.add(new Fruit("Orange", 180, 0.0));
* fruits.add(new Fruit("Mango", 140, 0));
* fruits.add(new Fruit("Grapes", 200, 0.0));
* fruits.add(new Fruit("Strawberry", 50, 0.0));
* fruits.add(new Fruit("Pineapple", 300, 0.0));
* fruits.add(new Fruit("Watermelon", 5000, 0.0));
* fruits.add(new Fruit("Kiwi", 80, 0.0));
* fruits.add(new Fruit("Peach", 200, 0.0));
* fruits.add(new Fruit("Pear", 180, 0.0));
* fruits.add(new Fruit("Cherry", 10, 0.0));
* fruits.add(new Fruit("Blueberry", 5, 0.0));
* fruits.add(new Fruit("Raspberry", 3, 0.0));
* fruits.add(new Fruit("Plum", 120, 0.0));
* fruits.add(new Fruit("Cantaloupe", 1200, 0.0));
* // Add more fruits...
* return fruits;
* }
* private static char gradeSweetnessKNN(Fruit targetFruit, List<Fruit> allFruits) {
* int k = 3; // Number of neighbors to consider
* // Find k-nearest neighbors
* List<FruitDistance> distances = new ArrayList<>();
* for (Fruit fruit : allFruits) {
* double distance = calculateDistance(targetFruit, fruit);
* distances.add(new FruitDistance(fruit, distance));
* }
* // Sort distances and get k-nearest neighbors
* Collections.sort(distances);
* List<Fruit> nearestNeighbors = new ArrayList<>();
* for (int i = 0; i < k; i++) {
* nearestNeighbors.add(distances.get(i).getFruit());
* }
* // Predict grade based on majority vote
* int excellentCount = 0;
* int goodCount = 0;
* int poorCount = 0;
* for (Fruit neighbor : nearestNeighbors) {
* char grade = gradeSweetnessRuleBased(neighbor.getSweetness());
* switch (grade) {
* case 'A':
* excellentCount++;
* break;
* case 'B':
* goodCount++;
* break;
* case 'C':
* poorCount++;
* break;
* }
* }
* // Return the majority vote as the predicted grade
* if (excellentCount >= goodCount && excellentCount >= poorCount) {
* return 'A'; // Excellent
* } else if (goodCount >= excellentCount && goodCount >= poorCount) {
* return 'B'; // Good
* } else {
* return 'C'; // Poor
* }
* }
* private static char gradeSweetnessRuleBased(double sweetness) {
* // Rule-based grading logic based on sweetness
* if (sweetness >= 6.0) {
* return 'A'; // Excellent
* } else if (sweetness >= 5.0) {
* return 'B'; // Good
* } else {
* return 'C'; // Poor
* }
* }
* private static double calculateDistance(Fruit fruit1, Fruit fruit2) {
* // Euclidean distance calculation between sweetness values
* return Math.abs(fruit1.getSweetness() - fruit2.getSweetness());
* }
* }
* class Fruit {
* private final String name;
* private final double weight;
* private double sweetness;
* public Fruit(String name, double weight, double sweetness) {
* this.name = name;
* this.weight = weight;
* this.sweetness = sweetness;
* }
* public String getName() {
* return name;
* }
* public double getWeight() {
* return weight;
* }
* public double getSweetness() {
* return sweetness;
* }
* public void setSweetness(double sweetness) {
* this.sweetness = sweetness;
* }
* }
* class FruitDistance implements Comparable<FruitDistance> {
* private final Fruit fruit;
* private final double distance;
* public FruitDistance(Fruit fruit, double distance) {
* this.fruit = fruit;
* this.distance = distance;
* }
* public Fruit getFruit() {
* return fruit;
* }
* @Override
* public int compareTo(FruitDistance other) {
* return Double.compare(this.distance, other.distance);
* }
* }

**Snapshots**

* Project Screenshots

