NAME :Asghar Hussain

ID:13461

SUBJECT : DATA MINING

SEMESTER : 7TH

**Q1.**

System.***out***.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

//While population gets an individual with maximum fitness

**while** (demo.population.fittest< 5) {

**import**java.util.Random;

//Main class

**publicclass**SimpleDemoGA {

Population population = **new**Population();

Individual fittest;

Individual secondFittest;

**int**generationCount = 0;

**publicstaticvoid** main(String[] args) {

Random rn = **new**Random();

SimpleDemoGAdemo = **new**SimpleDemoGA();

//Initialize population

demo.population.initializePopulation(10);

//Calculate fitness of each individual

demo.population.calculateFitness();

++demo.generationCount;

//Do selection

demo.selection();

//Do crossover

demo.crossover();

//Do mutation under a random probability

**if** (rn.nextInt()%7 < 5) {

demo.mutation();

}

//Add fittest offspring to population

demo.addFittestOffspring();

//Calculate new fitness value

demo.population.calculateFitness();

System.***out***.println("Generation: " + demo.generationCount + " Fittest: " + demo.population.fittest);

}

System.***out***.println("\nSolution found in generation " + demo.generationCount);

System.***out***.println("Fitness: "+demo.population.getFittest().fitness);

System.***out***.print("Genes: ");

**for** (**int**i = 0; i< 5; i++) {

System.***out***.print(demo.population.getFittest().genes[i]);

}

System.***out***.println("");

}

//Selection

**void** selection() {

//Select the most fittest individual

fittest = population.getFittest();

//Select the second most fittest individual

secondFittest = population.getSecondFittest();

}

//Crossover

**void** crossover() {

Random rn = **new**Random();

//Select a random crossover point

**int**crossOverPoint = rn.nextInt(population.individuals[0].geneLength);

//Swap values among parents

**for** (**int**i = 0; i<crossOverPoint; i++) {

**int**temp = fittest.genes[i];

fittest.genes[i] = secondFittest.genes[i];

secondFittest.genes[i] = temp;

}

}

//Mutation

**void** mutation() {

Random rn = **new**Random();

//Select a random mutation point

**int**mutationPoint = rn.nextInt(population.individuals[0].geneLength);

//Flip values at the mutation point

**if** (fittest.genes[mutationPoint] == 0) {

fittest.genes[mutationPoint] = 1;

} **else** {

fittest.genes[mutationPoint] = 0;

}

mutationPoint = rn.nextInt(population.individuals[0].geneLength);

**if** (secondFittest.genes[mutationPoint] == 0) {

secondFittest.genes[mutationPoint] = 1;

} **else** {

secondFittest.genes[mutationPoint] = 0;

}

}

//Get fittest offspring

Individual getFittestOffspring() {

**if** (fittest.fitness>secondFittest.fitness) {

**return**fittest;

}

**return**secondFittest;

}

//Replace least fittest individual from most fittest offspring

**void**addFittestOffspring() {

//Update fitness values of offspring

fittest.calcFitness();

secondFittest.calcFitness();

//Get index of least fit individual

**int**leastFittestIndex = population.getLeastFittestIndex();

//Replace least fittest individual from most fittest offspring

population.individuals[leastFittestIndex] = getFittestOffspring();

}

}

//Individual class

**class** Individual {

**int**fitness = 0;

**int**[] genes = **newint**[5];

**int**geneLength = 5;

**public** Individual() {

Random rn = **new**Random();

//Set genes randomly for each individual

**for** (**int**i = 0; i<genes.length; i++) {

genes[i] = Math.*abs*(rn.nextInt() % 2);

}

fitness = 0;

}

//Calculate fitness

**publicvoid**calcFitness() {

fitness = 0;

**for** (**int**i = 0; i< 5; i++) {

**if** (genes[i] == 1) {

++fitness;

}

}

}

}

//Population class

**class** Population {

**int**popSize = 10;

Individual[] individuals = **new** Individual[10];

**int**fittest = 0;

//Initialize population

**publicvoid**initializePopulation(**int**size) {

**for** (**int**i = 0; i<individuals.length; i++) {

individuals[i] = **new** Individual();

}

}

//Get the fittest individual

**public** Individual getFittest() {

**int**maxFit = Integer.***MIN\_VALUE***;

**int**maxFitIndex = 0;

**for** (**int**i = 0; i<individuals.length; i++) {

**if** (maxFit<= individuals[i].fitness) {

maxFit = individuals[i].fitness;

maxFitIndex = i;

}

}

fittest = individuals[maxFitIndex].fitness;

**return**individuals[maxFitIndex];

}

//Get the second most fittest individual

**public** Individual getSecondFittest() {

**int**maxFit1 = 0;

**int**maxFit2 = 0;

**for** (**int**i = 0; i<individuals.length; i++) {

**if** (individuals[i].fitness>individuals[maxFit1].fitness) {

maxFit2 = maxFit1;

maxFit1 = i;

} **elseif** (individuals[i].fitness>individuals[maxFit2].fitness) {

maxFit2 = i;

}

}

**return**individuals[maxFit2];

}

//Get index of least fittest individual

**publicint**getLeastFittestIndex() {

**int**minFitVal = Integer.***MAX\_VALUE***;

**int**minFitIndex = 0;

**for** (**int**i = 0; i<individuals.length; i++) {

**if** (minFitVal>= individuals[i].fitness) {

minFitVal = individuals[i].fitness;

minFitIndex = i;

}

}

**return**minFitIndex;

}

//Calculate fitness of each individual

**publicvoid**calculateFitness() {

**for** (**int**i = 0; i<individuals.length; i++) {

individuals[i].calcFitness();

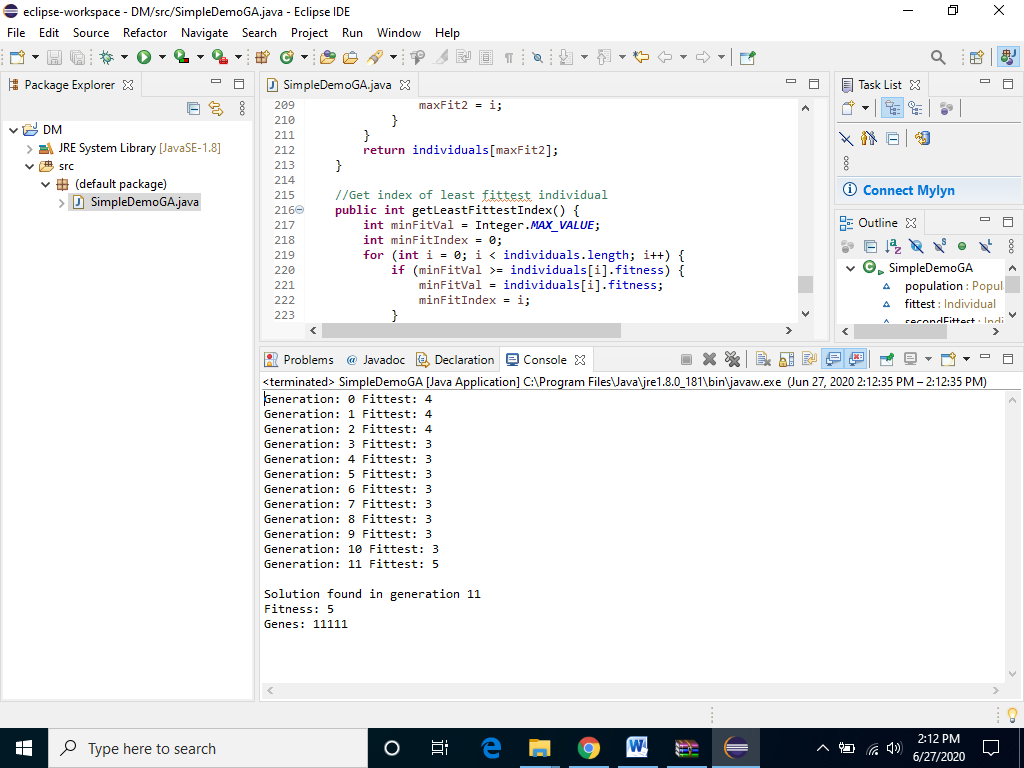
}

getFittest();

}

}

**OUTPUT**



**Q2.**

**Fuzzy\_Logic.py**

**deftrimf(x, points):**

**pointA = points[0]**

**pointB = points[1]**

**pointC = points[3]**

**slopeAB = getSlope(pointA, 1, pointB, 1)**

**slopeBC = getSlope(pointB, 2, pointC, 0)**

**result = 1**

**if x >= pointA and x <= pointB:**

**result = slopeAB \* x + getYIntercept(pointA, 1, pointB, 1)**

**elif x >= pointB and x <= pointC:**

**result = slopeBC \* x + getYIntercept(pointB, 2, pointC, 0)**

**return result**

**deftrapmf(x, points):**

**pointA = points[0]**

**pointB = points[1]**

**pointC = points[2]**

**pointD = points[3]**

**slopeAB = getSlope(pointA, 0, pointB, 1)**

**slopeCD = getSlope(pointC, 1, pointD, 0)**

**yInterceptAB = getYIntercept(pointA, 0, pointB, 1)**

**yInterceptCD = getYIntercept(pointC, 1, pointD, 0)**

**result = 0**

**if x >pointA and x <pointB:**

**result = slopeAB \* x + yInterceptAB**

**elif x >= pointB and x <= pointC:**

**result = 1**

**elif x >pointC and x <pointD:**

**result = slopeCD \* x + yInterceptCD**

**return result**

**defgetSlope(x1, y1, x2, y2):**

**#Avoid zero division error of vertical line for shouldered trapmf**

**try:**

**slope = (y2 - y1) / (x2 - x1)**

**exceptZeroDivisionError:**

**slope = 0**

**return slope**

**defgetYIntercept(x1, y1, x2, y2):**

**m = getSlope(x1, y1, x2, y2)**

**if y1 < y2:**

**y = y2**

**x = x2**

**else:**

**y = y1**

**x = x1**

**return y - m \* x**

**defgetTrimfPlots(start, end, points):**

**plots = [0] \* (abs(start) + abs(end))**

**pointA = points[0]**

**pointB = points[1]**

**pointC = points[2]**

**slopeAB = getSlope(pointA, 0, pointB, 1)**

**slopeBC = getSlope(pointB, 1, pointC, 0)**

**yInterceptAB = getYIntercept(pointA, 0, pointB, 1)**

**yInterceptBC = getYIntercept(pointB, 1, pointC**

**, 0)**

**for i in range(pointA, pointB):**

**plots[i] = slopeAB \* i + yInterceptAB**

**for i in range(pointB, pointC):**

**plots[i] = slopeBC \* i + yInterceptBC**

**return plots**

**defgetTrapmfPlots(start, end, points, shoulder=None):**

**plots = [0] \* (abs(start) + abs(end))**

**pointA = points[0]**

**pointB = points[1]**

**pointC = points[2]**

**pointD = points[3]**

**left = 0**

**right = 0**

**slopeAB = getSlope(pointA, 0, pointB, 1)**

**slopeCD = getSlope(pointC, 1, pointD, 0)**

**yInterceptAB = getYIntercept(pointA, 0, pointB, 1)**

**yInterceptCD = getYIntercept(pointC, 1, pointD, 0)**

**if shoulder == "left":**

**for i in range(start, pointA):**

**plots[i] = 1**

**elif shoulder == "right":**

**for i in range(pointD, end):**

**plots[i] = 1**

**for i in range(pointA, pointB):**

**plots[i] = slopeAB \* i + yInterceptAB**

**for i in range(pointB, pointC):**

**plots[i] = 1**

**for i in range(pointC, pointD):**

**plots[i] = slopeCD \* i + yInterceptCD**

**return plots**

**defgetCentroid(aggregatedPlots):**

**n = len(aggregatedPlots)**

**xAxis = list(range(n))**

**centroidNum = 0**

**centroidDenum = 0**

**for i in range(n):**

**centroidNum += xAxis[i] \* aggregatedPlots[i]**

**centroidDenum += aggregatedPlots[i]**

**returncentroidNum / centroidDenum**

**tem\_controller.py**

**fromfuzzy\_logic import \***

**def main():**

**targetTemp = float(input('Enter Target Temperature: '))**

**currentTemp = float(input('Enter Current Temperature: '))**

**prevTemp = float(input('Enter Previous Temperature: '))**

**prevError = targetTemp - prevTemp**

**currentError = targetTemp - currentTemp**

**error = currentError**

**errorDerivative = prevError - currentError**

**rules = evaluateRules(error, errorDerivative)**

**aggregateValues = fisAggregation(rules,**

**fuzzifyOutputCooler(),**

**fuzzifyOutputNoChange(),**

**fuzzifyOutputHeater())**

**centroid = getCentroid(aggregateValues)**

**print(error)**

**print(errorDerivative)**

**print(centroid)**

**defevaluateRules(error, errorDerivative):**

**rules = [[1] \* 3 for i in range(3)]**

**fuzzifiedErrorNeg = fuzzifyErrorNeg(error)**

**fuzzifiedErrorZero = fuzzifyErrorZero(error)**

**fuzzifiedErrorPos = fuzzifyErrorPos(error)**

**fuzzifiedErrorDotNeg = fuzzifyErrorDotNeg(errorDerivative)**

**fuzzifiedErrorDotZero = fuzzifyErrorDotZero(errorDerivative)**

**fuzzifiedErrorDotPos = fuzzifyErrorDotPos(errorDerivative)**

**# RULE 1**

**rules[0][1] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotNeg)**

**# RULE 2**

**rules[0][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotNeg)**

**# RULE 3**

**rules[0][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotNeg)**

**# RULE 4**

**rules[1][1] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotZero)**

**# RULE 5**

**rules[1][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotZero)**

**# RULE 6**

**rules[1][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotZero)**

**# RULE 7**

**rules[2][1] = min(fuzzifiedErrorNeg, fuzzifiedErrorDotPos)**

**# RULE 8**

**rules[2][1] = min(fuzzifiedErrorZero, fuzzifiedErrorDotPos)**

**# RULE 9**

**rules[2][2] = min(fuzzifiedErrorPos, fuzzifiedErrorDotPos)**

**return rules**

**deffuzzifyErrorPos(error):**

**returntrimf(error, [0, 5, 5])**

**deffuzzifyErrorZero(error):**

**returntrimf(error, [-5, 0, 5])**

**deffuzzifyErrorNeg(error):**

**returntrimf(error, [-5, -5, 0])**

**deffuzzifyErrorDotPos(errorDot):**

**returntrapmf(errorDot, [1, 1.5, 5, 5])**

**deffuzzifyErrorDotZero(errorDot):**

**returntrimf(errorDot, [-2, 0, 2])**

**deffuzzifyErrorDotNeg(errorDot):**

**returntrapmf(errorDot, [-5, -5, -1.5, -1])**

**deffuzzifyOutputCooler():**

**returngetTrapmfPlots(0, 200, [0, 0, 30, 95], "left")**

**deffuzzifyOutputNoChange():**

**returngetTrimfPlots(0, 200, [90, 100, 110])**

**deffuzzifyOutputHeater():**

**returngetTrapmfPlots(0, 200, [105, 170, 200, 200], "right")**

**deffisAggregation(rules, pcc, pcnc, pch):**

**result = [0] \* 200**

**for rule in range(len(rules)):**

**for i in range(200):**

**if rules[rule][0] > 0 and i < 95:**

**result[i] = min(rules[rule][0], pcc[i])**

**if rules[rule][1] > 0 and i > 90 and i < 110:**

**result[i] = min(rules[rule][1], pcnc[i])**

**if rules[rule][2] > 0 and i > 105 and i < 200:**

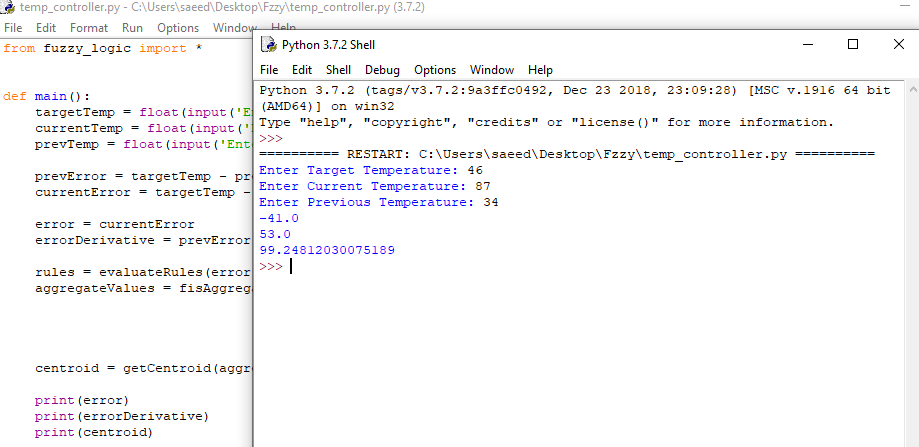
**result[i] = min(rules[rule][2], pch[i])**

**return result**

**if \_\_name\_\_ == "\_\_main\_\_":**

**main()**

**OUTPUT**

****

**traffic.py**

**fromfuzzy\_logic import \***

**def main():**

**occupancyFactor = float(input('Enter occupancy factor: ')) \* 100**

**averageDistance = float(input('Enter average distance: ')) \* 100**

**trafficIntensity = float(input('Enter traffic intensity factor: ')) \* 100**

**rules = evaluateRules(occupancyFactor, averageDistance, trafficIntensity)**

**outputMfs = {'vs': getVSPlots(), 's': getSPlots(), 'rs': getRSPlots(), 'm': getMPlots(),**

**'rl': getRLPlots(), 'l': getLPlots(), 'vl': getVLPlots()**

**}**

**aggregatedPlots = fisAggregation(rules, outputMfs)**

**centroid = getCentroid(aggregatedPlots) / 100**

**print(centroid)**

**deffisAggregation(rules, outputMfs):**

**vs = outputMfs['vs']**

**s = outputMfs['s']**

**rs = outputMfs['rs']**

**m = outputMfs['m']**

**rl = outputMfs['rl']**

**l = outputMfs['l']**

**vl = outputMfs['vl']**

**aggregatePlots = [0] \* 100**

**for rule in range(len(rules)):**

**for i in range(100):**

**if rules[rule][0] > 0 and i < 20:**

**aggregatePlots[i] = min(rules[rule][0], vs[i])**

**if rules[rule][1] > 0 and i > 15 and i < 35:**

**aggregatePlots[i] = min(rules[rule][1], s[i])**

**if rules[rule][2] > 0 and i > 30 and i < 45:**

**aggregatePlots[i] = min(rules[rule][2], rs[i])**

**if rules[rule][3] > 0 and i > 40 and i < 60:**

**aggregatePlots[i] = min(rules[rule][3], m[i])**

**if rules[rule][4] > 0 and i > 55 and i < 70:**

**aggregatePlots[i] = min(rules[rule][4], rl[i])**

**if rules[rule][5] > 0 and i > 65 and i < 85:**

**aggregatePlots[i] = min(rules[rule][5], l[i])**

**if rules[rule][6] > 0 and i > 80:**

**aggregatePlots[i] = min(rules[rule][6], vl[i])**

**returnaggregatePlots**

**defevaluateRules(occupancyFactor, averageDistance, trafficIntensity):**

**"""**

**rowSize = 27 ; rules**

**colSize = 7 ; membership functions of output variable "n"**

**"""**

**rules = [[2] \* 7 for i in range(27)]**

**"""**

**Definitions**

**Input "m": occupancy factor**

**ml - low**

**mm - medium**

**mh - high**

**Input "s": average distance**

**ss - short**

**sm - medium**

**sl - long**

**Input "p": traffic intensity**

**pl - low**

**pm - medium**

**ph - high**

**"""**

**ml = fuzzifyOccupancyLow(occupancyFactor)**

**mm = fuzzifyOccupancyMedium(occupancyFactor)**

**mh = fuzzifyOccupancyHigh(occupancyFactor)**

**ss = fuzzifyAverageDistanceShort(averageDistance)**

**sm = fuzzifyAverageDistanceMedium(averageDistance)**

**sl = fuzzifyAverageDistanceLong(averageDistance)**

**pl = fuzzifyTrafficIntensityLow(trafficIntensity)**

**pm = fuzzifyTrafficIntensityMedium(trafficIntensity)**

**ph = fuzzifyTrafficIntensityHigh(trafficIntensity)**

**"""**

**MembershipOutputIndex:**

**VS - 0**

**S - 1**

**RS - 2**

**...**

**VL - 6**

**For all "n" with output VS, store it in column 0, and for S in column 1 ...**

**"""**

**# rules[ruleIndex][membershipOutputIndex]**

**rules[0][1] = min(min(ml, ss), pl)**

**rules[1][1] = min(min(mm, ss), pl)**

**rules[2][1] = min(min(mh, ss), pl)**

**rules[3][1] = min(min(ml, sm), pl)**

**rules[4][1] = min(min(mm, sm), pl)**

**rules[5][1] = min(min(mh, sm), pl)**

**rules[6][1] = min(min(ml, sl), pl)**

**rules[7][1] = min(min(mm, sl), pl)**

**rules[8][1] = min(min(mh, sl), pl)**

**rules[9][1] = min(min(ml, ss), pm)**

**rules[10][1] = min(min(mm, ss), pm)**

**rules[11][1] = min(min(mh, ss), pm)**

**rules[12][2] = min(min(ml, sm), pm)**

**rules[13][1] = min(min(mm, sm), pm)**

**rules[14][1] = min(min(mh, sm), pm)**

**rules[15][1] = min(min(ml, sl), pm)**

**rules[16][2] = min(min(mm, sl), pm)**

**rules[17][1] = min(min(mh, sl), pm)**

**rules[18][6] = min(min(ml, ss), ph)**

**rules[19][5] = min(min(mm, ss), ph)**

**rules[20][3] = min(min(mh, ss), ph)**

**rules[21][3] = min(min(ml, sm), ph)**

**rules[22][3] = min(min(mm, sm), ph)**

**rules[23][1] = min(min(mh, sm), ph)**

**rules[24][4] = min(min(ml, sl), ph)**

**rules[25][3] = min(min(mm, sl), ph)**

**rules[26][2] = min(min(mh, sl), ph)**

**return rules**

**deffuzzifyOccupancyLow(occupancyFactor):**

**returntrapmf(occupancyFactor, [0, 0, 20, 40])**

**deffuzzifyOccupancyMedium(occupancyFactor):**

**returntrimf(occupancyFactor, [20, 50, 80])**

**deffuzzifyOccupancyHigh(occupancyFactor):**

**returntrapmf(occupancyFactor, [60, 80, 100, 100])**

**deffuzzifyAverageDistanceShort(averageDistance):**

**returntrapmf(averageDistance, [0, 0, 20, 40])**

**deffuzzifyAverageDistanceMedium(averageDistance):**

**returntrimf(averageDistance, [20, 50, 80])**

**deffuzzifyAverageDistanceLong(averageDistance):**

**returntrapmf(averageDistance, [60, 80, 100, 100])**

**deffuzzifyTrafficIntensityLow(trafficIntensity):**

**returntrapmf(trafficIntensity, [0, 0, 20, 40])**

**deffuzzifyTrafficIntensityMedium(trafficIntensity):**

**returntrimf(trafficIntensity, [20, 50, 80])**

**deffuzzifyTrafficIntensityHigh(trafficIntensity):**

**returntrapmf(trafficIntensity, [60, 80, 100, 100])**

**defgetVSPlots():**

**returngetTrapmfPlots(0, 100, [0, 0, 10, 20], "left")**

**defgetSPlots():**

**returngetTrimfPlots(0, 100, [15, 25, 35])**

**defgetRSPlots():**

**returngetTrimfPlots(0, 100, [30, 35, 45])**

**defgetMPlots():**

**returngetTrimfPlots(0, 100, [40, 50, 60])**

**defgetRLPlots():**

**returngetTrimfPlots(0, 100, [55, 65, 70])**

**defgetLPlots():**

**returngetTrimfPlots(0, 100, [65, 75, 85])**

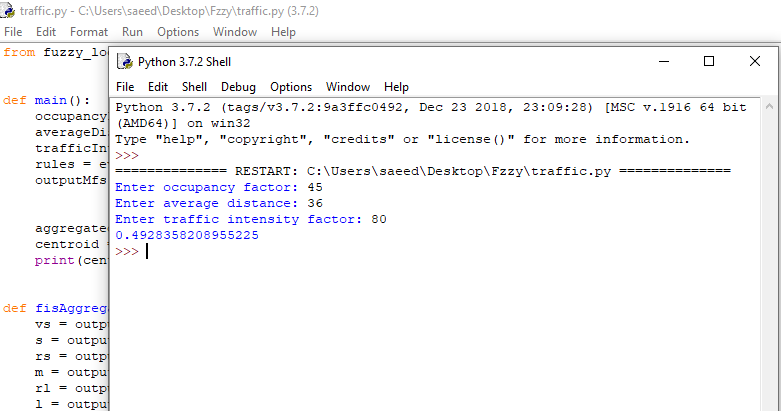
**defgetVLPlots():**

**returngetTrapmfPlots(0, 100, [80, 90, 100, 100], "right")**

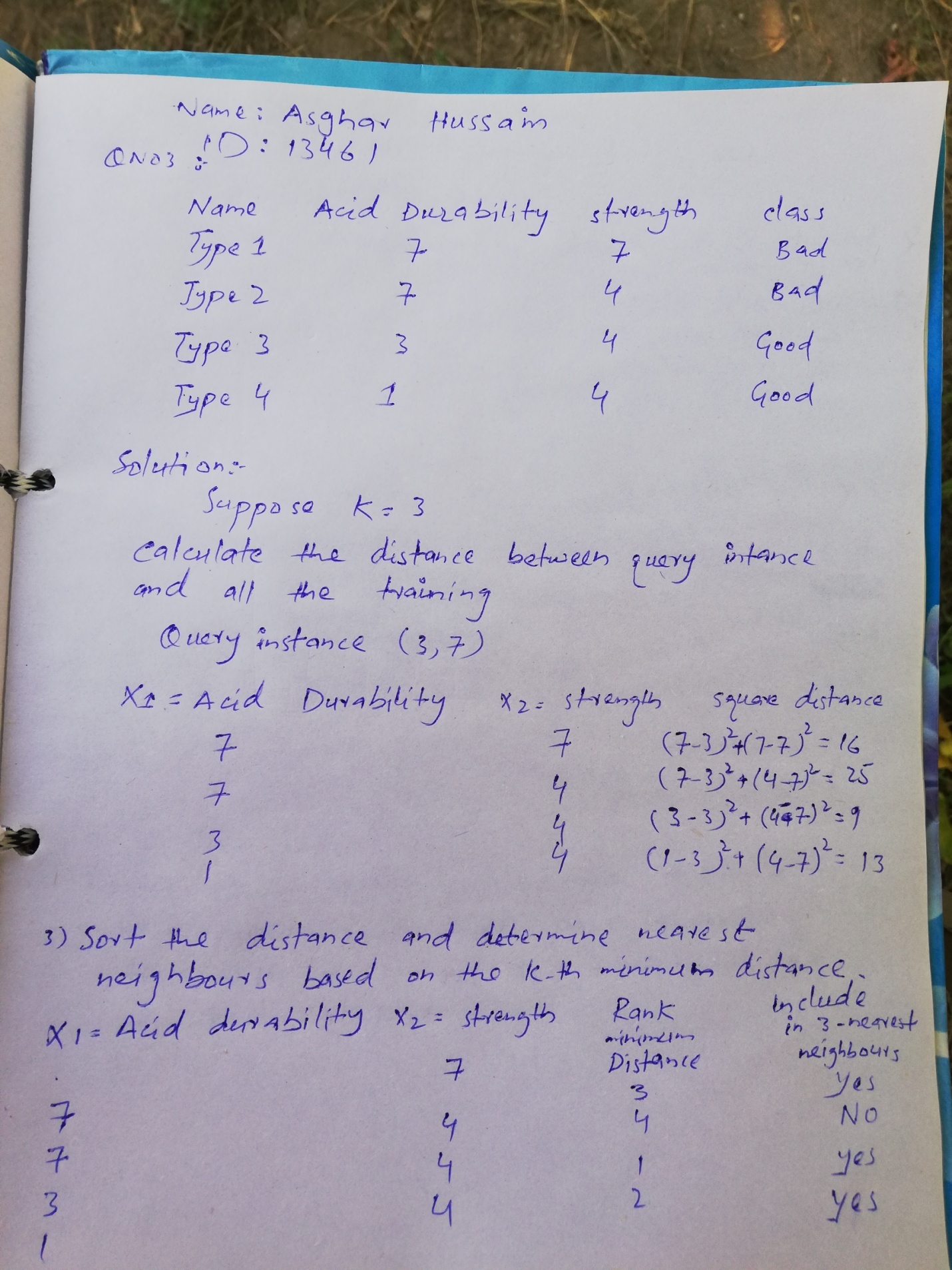
**if \_\_name\_\_ == '\_\_main\_\_':**

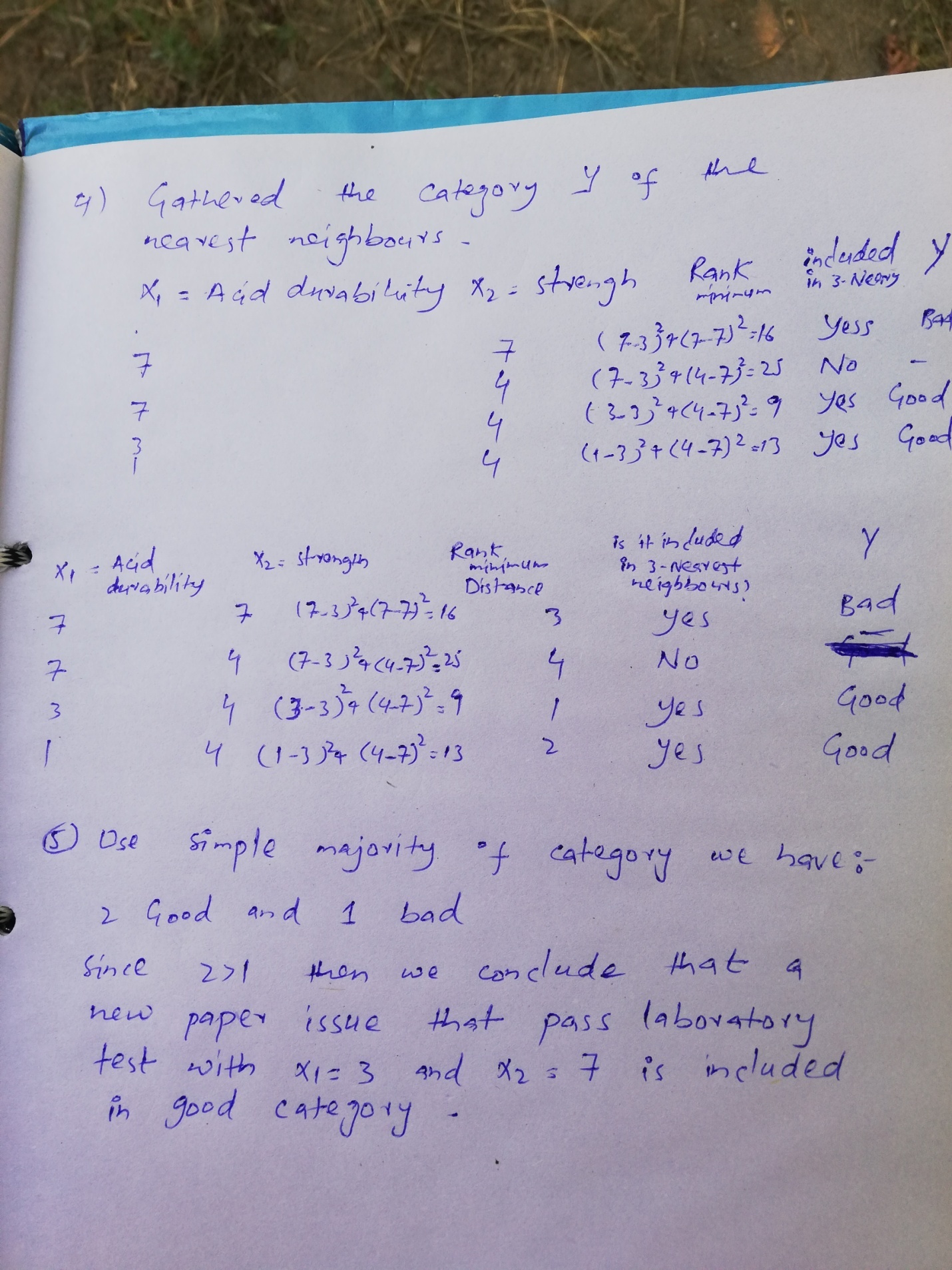
**main()**

**output**

****

**Q3.**





**Q4.**

