**Name: Syed Daniyal Shah**

**ID: 14127**

**Paper: Data Mining**

**Date: 27/6/2020**

Q1. Implement a code of Genetic Algorithm in any language and show the output. (10)

# 

# #include <bits/stdc++.h>

# using namespace std;

# 

# #define POPULATION\_SIZE 100

# 

# const string GENES = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP"\

# "QRSTUVWXYZ 1234567890, .-;:\_!\"#%&/()=?@${[]}";

# 

# const string TARGET = "I love GeeksforGeeks";

# 

# int random\_num(int start, int end)

# {

# int range = (end-start)+1;

# int random\_int = start+(rand()%range);

# return random\_int;

# }

# char mutated\_genes()

# {

# int len = GENES.size();

# int r = random\_num(0, len-1);

# return GENES[r];

# }

# 

# string create\_gnome()

# {

# int len = TARGET.size();

# string gnome = "";

# for(int i = 0;i<len;i++)

# gnome += mutated\_genes();

# return gnome;

# }

# 

# class Individual

# {

# public:

# string chromosome;

# int fitness;

# Individual(string chromosome);

# Individual mate(Individual parent2);

# int cal\_fitness();

# };

# 

# Individual::Individual(string chromosome)

# {

# this->chromosome = chromosome;

# fitness = cal\_fitness();

# };

# 

# Individual Individual::mate(Individual par2)

# {

# // chromosome for offspring

# string child\_chromosome = "";

# 

# int len = chromosome.size();

# for(int i = 0;i<len;i++)

# {

# // random probability

# float p = random\_num(0, 100)/100;

# 

# // if prob is less than 0.45, insert gene

# // from parent 1

# if(p < 0.45)

# child\_chromosome += chromosome[i];

# 

# // if prob is between 0.45 and 0.90, insert

# // gene from parent 2

# else if(p < 0.90)

# child\_chromosome += par2.chromosome[i];

# 

# else

# child\_chromosome += mutated\_genes();

# }

# 

# return Individual(child\_chromosome);

# };

# 

# int Individual::cal\_fitness()

# {

# int len = TARGET.size();

# int fitness = 0;

# for(int i = 0;i<len;i++)

# {

# if(chromosome[i] != TARGET[i])

# fitness++;

# }

# return fitness;

# };

# 

# bool operator<(const Individual &ind1, const Individual &ind2)

# {

# return ind1.fitness < ind2.fitness;

# }

# 

# int main()

# {

# srand((unsigned)(time(0)));

# 

# // current generation

# int generation = 0;

# 

# vector<Individual> population;

# bool found = false;

# 

# // create initial population

# for(int i = 0;i<POPULATION\_SIZE;i++)

# {

# string gnome = create\_gnome();

# population.push\_back(Individual(gnome));

# }

# 

# while(! found)

# {

# sort(population.begin(), population.end());

# 

# if(population[0].fitness <= 0)

# {

# found = true;

# break;

# }

# vector<Individual> new\_generation;

# 

# int s = (10\*POPULATION\_SIZE)/100;

# for(int i = 0;i<s;i++)

# new\_generation.push\_back(population[i]);

# s = (90\*POPULATION\_SIZE)/100;

# for(int i = 0;i<s;i++)

# {

# int len = population.size();

# int r = random\_num(0, 50);

# Individual parent1 = population[r];

# r = random\_num(0, 50);

# Individual parent2 = population[r];

# Individual offspring = parent1.mate(parent2);

# new\_generation.push\_back(offspring);

# }

# population = new\_generation;

# cout<< "Generation: " << generation << "\t";

# cout<< "String: "<< population[0].chromosome <<"\t";

# cout<< "Fitness: "<< population[0].fitness << "\n";

# 

# generation++;

# }

# cout<< "Generation: " << generation << "\t";

# cout<< "String: "<< population[0].chromosome <<"\t";

# cout<< "Fitness: "<< population[0].fitness << "\n";

# } **Output:**

# Generation: 1 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

# Generation: 2 String: tO{"-?=jH[k8=B4]Oe@} Fitness: 18

# Generation: 3 String: .#lRWf9k\_Ifslw #O$k\_ Fitness: 17

# Generation: 4 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

# Generation: 5 String: .-1Rq?9mHqk3Wo]3rek\_ Fitness: 16

# Generation: 6 String: A#ldW) #lIkslw cVek) Fitness: 14

# Generation: 7 String: A#ldW) #lIkslw cVek) Fitness: 14

# Generation: 8 String: (, o x \_x%Rs=, 6Peek3 Fitness: 13

# .

# .

# .

# Generation: 29 String: I lope Geeks#o, Geeks Fitness: 3

# Generation: 30 String: I loMe GeeksfoBGeeks Fitness: 2

# Generation: 31 String: I love Geeksfo0Geeks Fitness: 1

# Generation: 32 String: I love Geeksfo0Geeks Fitness: 1

# Generation: 33 String: I love Geeksfo0Geeks Fitness: 1

# Generation: 34 String: I love GeeksforGeeks Fitness: 0

Q2. Implement a code of Fuzzy logic in any language and show the output. (10)

Answer:

**Code :**

#include <iostream>

#include <conio.h>

#include <cmath>

#include <cstring>

const double cdMinimumPrice =0;

const double cdMaximumPrice =70;

using namespace std;

class CFuzzyFunction

{

protected :

double dLeft, dRight;

char cType;

char\* sName;

public:

CFuzzyFunction(){};

virtual ~CFuzzyFunction(){ delete [] sName; sName=NULL;}

virtual void

setInterval(double l,

double r)

{dLeft=l; dRight=r;}

virtual void

setMiddle( double dL=0,

double dR=0)=0;

virtual void

setType(char c)

{ cType=c;}

virtual void

setName(const char\* s)

{

sName = new char[strlen(s)+1];

strcpy(sName,s);

}

bool

isDotInInterval(double t)

{

if((t>=dLeft)&&(t<=dRight)) return true; else return false;

}

char getType(void)const{ return cType;}

void

getName() const

{

cout<<sName<<endl;

}

virtual double getValue(double t)=0;

};

class CTriangle : public CFuzzyFunction

{

private:

double dMiddle;

public:

void

setMiddle(double dL, double dR)

{

dMiddle=dL;

}

double

getValue(double t)

{

if(t<=dLeft)

return 0;

else if(t<dMiddle)

return (t-dLeft)/(dMiddle-dLeft);

else if(t==dMiddle)

return 1.0;

else if(t<dRight)

return (dRight-t)/(dRight-dMiddle);

else

return 0;

}

};

class CTrapezoid : public CFuzzyFunction

{

private:

double dLeftMiddle, dRightMiddle;

public:

void

setMiddle(double dL, double dR)

{

dLeftMiddle=dL; dRightMiddle=dR;

}

double

getValue(double t)

{

if(t<=dLeft)

return 0;

else if(t<dLeftMiddle)

return (t-dLeft)/(dLeftMiddle-dLeft);

else if(t<=dRightMiddle)

return 1.0;

else if(t<dRight)

return (dRight-t)/(dRight-dRightMiddle);

else

return 0;

}

};

int

main(void)

{

CFuzzyFunction \*FuzzySet[3];

FuzzySet[0] = new CTrapezoid;

FuzzySet[1] = new CTriangle;

FuzzySet[2] = new CTrapezoid;

FuzzySet[0]->setInterval(-5,30);

FuzzySet[0]->setMiddle(0,20);

FuzzySet[0]->setType('r');

FuzzySet[0]->setName("low\_price");

FuzzySet[1]->setInterval(25,45);

FuzzySet[1]->setMiddle(35,35);

FuzzySet[1]->setType('t');

FuzzySet[1]->setName("good\_price");

FuzzySet[2]->setInterval(40,75);

FuzzySet[2]->setMiddle(50,70);

FuzzySet[2]->setType('r');

FuzzySet[2]->setName("to\_expensive");

double dValue;

do

{

cout<<"\nImput the value->"; cin>>dValue;

if(dValue<cdMinimumPrice) continue;

if(dValue>cdMaximumPrice) continue;

for(int i=0; i<3; i++)

{

cout<<"\nThe dot="<<dValue<<endl;

if(FuzzySet[i]->isDotInInterval(dValue))

cout<<"In the interval";

else

cout<<"Not in the interval";

cout<<endl;

cout<<"The name of function is"<<endl;

FuzzySet[i]->getName();

cout<<"and the membership is=";

cout<<FuzzySet[i]->getValue(dValue);

}

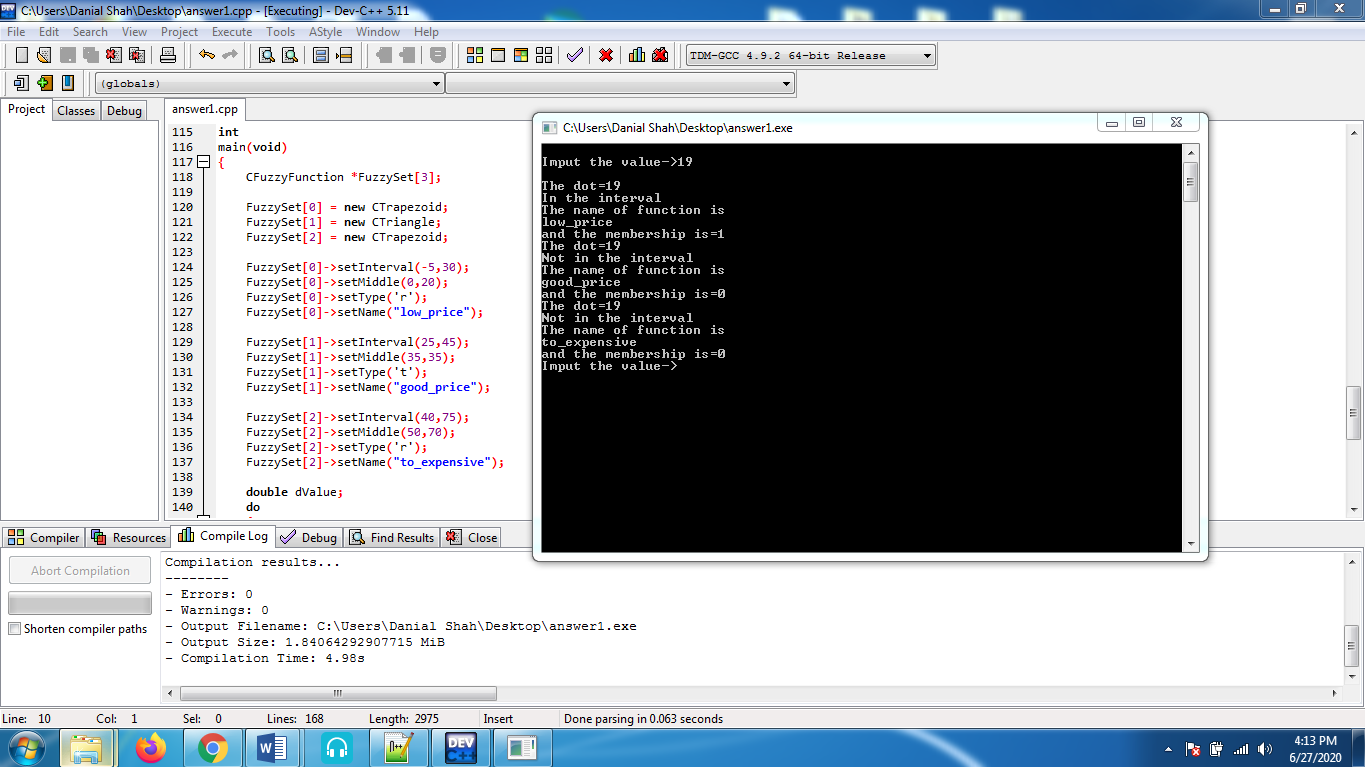
}

while(true);

getch();

}

Output:



Q3. Solve this using KNN. (15)

Answer:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Acid Durability | Strength | Class |
| Type-A | 7 | 7 | Bad |
| Type-B | 7 | 4 | Bad |
| Type-C | 3 | 4 | Good |
| Type-D | 1 | 4 | Good |

Text Data Acid durability = 3

Strength=7, Class =?  
  
Similarity = Calculate using measure like Euclidean.

d(p1q) = d (q1p) =

=

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | Acid Durability | Strength | Class | Distance |
| Type-A | 7 | 7 | Bad | Sqrt((7-3)²+(7-7)²)=4 |
| Type-B | 7 | 4 | Bad | 5 |
| Type-C | 3 | 4 | Good | 3 |
| Type-D | 1 | 4 | Good | 3.4 |

Rank these attributes

We can Rank them according to Minimum Distance.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| NAME | Acid Durability | Strength | Class | Distance | Rank |
| Type-A | 7 | 7 | Bad | 4 | 3 |
| Type-B | 7 | 4 | Bad | 5 | 4 |
| Type-C (if k =1) | 3 | 4 | Good | 3 | 1 |
| Type-D (if k = 2) | 1 | 4 | Good | 3.6 | 2 |

Now if k = 1

Then Type-C will be its immediate neighbor, good.

If k = 2

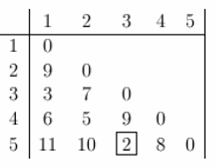
Then Type-D and Type-C both will be immediate neighbors based on two neighbor.

Q4. Give solved example of hierarchical Clustering. (15)

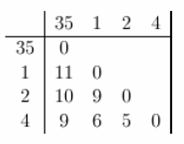
Answer:

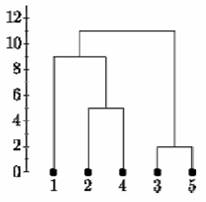
**Example of Complete Linkage Clustering**

Gathering starts by calculating a partition between each pair of units that you have to pack. A division grid will be symmetric (considering the way that the partition among x and y is proportionate to the detachment among y and x) and will have zeroes on the corner to corner (in light of the fact that every thing is partition zero from itself). The table underneath is an instance of a partition structure. Simply the lower triangle is showed up, in light of the fact that the upper triangle can be filled in by reflection.



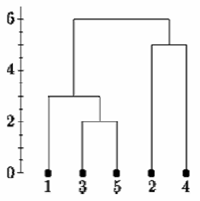
Presently lets begin grouping. The littlest separation is somewhere in the range of three and five and they get connected up or combined first into a the bunch '35'.

To obtain the new division grid, we need to clear the 3 and 5 areas, and displace it by an entry "35" . Since we are using completed linkage gathering, the division among "35" and each other thing is the restriction of the partition between this thing and 3 and this thing and 5. For example, d(1,3)= 3 and d(1,5)=11. Thusly, D(1,"35")=11. This gives us the new division structure. The things with the smallest division get gathered immediately. This will be 2 and 4.

Continuing thusly, after 6 phases, everything is packed. This is summarized underneath. On this plot, the y-turn shows the division between the articles at the time they were gathered. This is known as the bundle stature. Different portrayals use different extents of pack stature.

Complete Linkage

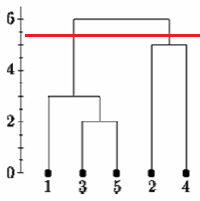
Below is the single linkage dendrogram for the same distance matrix. It starts with cluster "35" but the distance between "35" and each item is now the minimum of d(x,3) and d(x,5). So c(1,"35")=3.



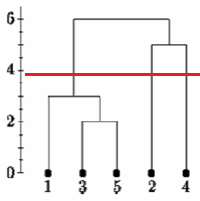
Single Linkage

**Determining clusters**

One of the issues with various leveled grouping is that there is no target approach to state what number of bunches there are.

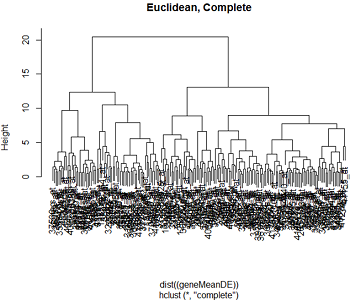
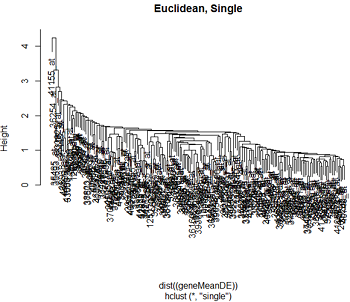
In the event that we cut the single linkage tree at the point demonstrated as follows, we would state that there are two bunches

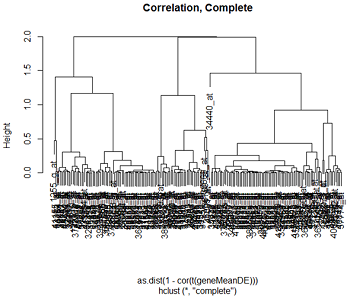
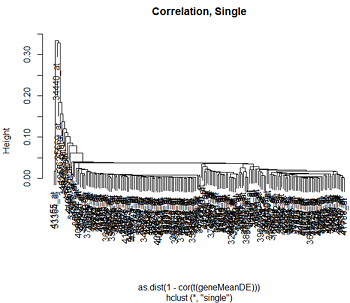
However, if we cut the tree lower we might say that there is one cluster and two singletons.



There is no generally settled upon way to deal with pick where to cut the tree. We should look at some certified data. In homework 5 we consider quality enunciation in 4 locale of 3 human and 3 chimpanzee minds. The RNA was hybridized to Affymetrix human quality verbalization microarrays. We normalized the data using RMA and did a differential verbalization assessment using LIMMA. Here we picked the 200 most out and out differentially imparted characteristics from the assessment. We bunch all the differentially imparted characteristics reliant on their mean verbalization in all of the 8 creature classes by mind territory drugs

Here are the packs subject to Euclidean partition and relationship division, using aggregate and single linkage gathering..

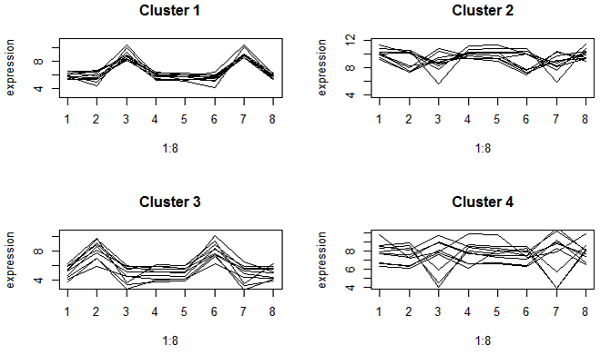
 

**We can see that the gathering structure for complete linkage partition will all in all make limited lots of packs, while single linkage will all in all remember each point for go to the gathering, making long cheap bundles. As we would foresee from our discussion of detachments, Euclidean division and association partition produce through and through various dendrograms.**

**Dynamic gathering doesn't unveil to us what number of bundles there are, or where to cut the dendrogram to shape packs. In R there is a limit cutttree which will cut a tree into bundles at a predefined height. In any case, considering our portrayal, we might need to cut the long branches at different heights. Notwithstanding, there is a respectable proportion of subjectivity in making sense of which branches should and should not be cut to outline separate gatherings.Understanding the clusters**

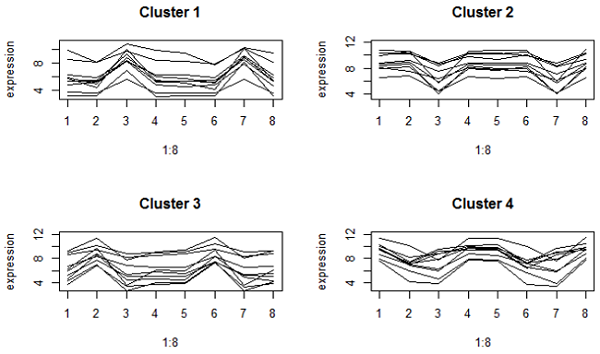
To grasp the gatherings we generally speaking plot the log2(expression) estimations of the characteristics in the pack, or toward the day's end, plot the quality enunciations over the models. (The numbering in these charts are totally abstract.) Even anyway the medications are unordered, I generally speaking interface the centers beginning from a lone part to make the model all the more clear. These are called profile plots.

Here is a bit of the profile plots from complete linkage packing when we used Euclidean partition:



These look firmly pressed. Nonetheless, bunches 2 and 4 have qualities with various here and there designs, since they have about a similar mean articulation. Group 2 are exceptionally communicated qualities.

Here's what we got when we use correlation distance:



These are significantly looser on the y-rotate since relationship bases on the enunciation plan, not the mean. Regardless, all the characteristics in a comparative gathering have a zenith or valley in comparative drugs (which are cerebrum territories by species blends). Gatherings 1 and 2 are characteristics that are independently higher or lower in the cerebellum diverged from other cerebrum districts in the two species.

Selecting a gene list

On a crucial level it is possible to pack all the characteristics, in spite of the way that envisioning an enormous dendrogram might be dubious. For the most part, a preliminary assessment, for instance, differential enunciation examination is used to pick characteristics for gathering. There are legitimate legitimizations to do all things considered, regardless of the way that there are furthermore a couple of cautions.

Routinely in quality enunciation, the partition metric used is relationship division. Relationship partition is comparable to centering and scaling the data, and a short time later using Euclidean detachment. When there are efficient treatment impacts, we expect the variability of value enunciation from treatment to treatment to be a mix of exact treatment effects and upheaval. When there are no treatment impacts, the capriciousness of value enunciation is simply because of disturbance. In any case, centering and scaling the data puts all variabity on a comparable scale. In this manner characteristics that show a model in view of chance are not discernable from those that have an exact part.

As we have seen, association partition has favored characteristic comprehension over Euclidean division for quality verbalization considers, anyway a comparative scaling that makes it supportive for finding naturally significant instances of value rule presents misleading results for characteristics that don't differentially impart. Picking characteristics subject to differential explanation examination clears characteristics which are likely going to have simply plausibility plans. This should overhaul the models found in the quality bundles.

As a counsel, nevertheless, think about the effects of value assurance on bundling tests or drugs. The picked characteristics are those which test positive in differential enunciation examination. Usage of those characteristics to bundle tests is uneven towards gathering the models by treatment.