CPE 261456 Introduction to Computational Intelligence

Computer Assignment 3 (Genetic Algorithm)

จัดทำโดย

นาย ณัชพล เพทายเทียมทอง

รหัสนักศึกษา 580610633

เสนอ

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

ภาควิชา วิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเชียงใหม่ ปีการศึกษา 2561

Genetic Algorithm

การทำงานของโปรแกรม

ทำการทดลองกับ Dataset Wisconsin Diagnostic Breast Cancer (WDBC) จาก UCI Machine learning Repository โดยจะมี 2 class และ 30 features

เริ่มต้นนั้นทำการอ่านค่าจาก Dataset มาเก็บไว้ใน Array ของโปรแกรมโดยตัว class ของ Dataset นี้นั้นรับมาเป็น string ดังนั้นจึงต้องทำการแปลงเป็นตัวเลขก่อนคือ ให้ class “M” มีค่าเท่ากับ 0และ class “B” มีค่าเท่ากับ 1 จากนั้นทำการกำหนดโครงสร้างของ Neural Network เมื่อได้โครงสร้างของ Neural Network มาแล้วกำหนดจำนวน Chromosome และทำการ initial Chromosome โดยใน Chromosome นั้นจะมี gene ที่ทำหน้าทีเก็บ weight ของโครงสร้าง Neural Network ซึ่งจะมีจำนวนตามโครงสร้างของ Neural Network โดยจะทำการสุ่มค่า weight ให้อยู่ในช่วงที่กำหนดไว้ในการทดสอบนี้ให้เป็น -1 ถึง 1จากนั้นก็สุ่มเสือก Chromosome มา 30 % เพื่อ Crossover ซึ่งในการทดลองจะใช้ one-point crossover โดยจะ crossover ตรงกลางของ Chromosome แล้วทำการหาค่า fitness แล้วนำ Chromosome Crossover กลับไปแทนที่กับ Chromosome ชุดก่อน crossover จะได้ set ของ Chromosome ชุดใหม่จากนั้นก็นำ 30 % มาทำการ mutation โดยจะสุ่มอีก 30 % เพื่อทำการ mutate โดยสุ่มเพิ่มค่าแต่จะไม่ให้เกินช่วง -1 ถึง 1 แล้วทำการหาค่า fitness แล้วนำมาแทนที่กับ Chromosome ชุดก่อน mutate จะได้ Chromosome โดยจะทำอย่างนี้ทุก Generation และจะหยุดหาเมื่อถึง Generation ที่กำหนด จากนั้นก็จะทำ Chromosome ของ Generation สุดท้ายมา test

การทดลอง

การทดลองที่ 1 ให้โครงสร้างของ Neural Network เป็น 30-15-5-2 และจำนวน Generation เป็น 200 จำนวนโครโมโซมเป็น 50

----------Fold: 1 ----------

Gen 200: 93.37231968810916

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 2 ----------

Gen 200: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 3 ----------

Gen 200: 92.78752436647173

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 4 ----------

Gen 200: 93.37231968810916

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 5 ----------

Gen 200: 91.61793372319688

---------- Start Test ----------

Test Accuracy: 87.5

----------Fold: 6 ----------

Gen 200: 92.20272904483431

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 7 ----------

Gen 200: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 8 ----------

Gen 200: 93.37231968810916

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 9 ----------

Gen 200: 92.78752436647173

---------- Start Test ----------

Test Accuracy: 87.5

----------Fold: 10 ----------

Gen 200: 93.37231968810916

---------- Start Test ----------

Test Accuracy: 94.64285714285714

Average Accuracy: 90.89285714285714

การทดลองที่ 2 ให้โครงสร้างของ Neural Network เป็น 30-30-10-2 และจำนวน Generation เป็น 200 จำนวนโครโมโซมเป็น 50

----------Fold: 1 ----------

Gen 200: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 2 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 3 ----------

Gen 200: 92.78752436647173

---------- Start Test ----------

Test Accuracy: 87.5

----------Fold: 4 ----------

Gen 200: 92.39766081871345

---------- Start Test ----------

Test Accuracy: 91.07142857142857

----------Fold: 5 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 6 ----------

Gen 200: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 87.5

----------Fold: 7 ----------

Gen 200: 92.5925925925926

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 8 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 9 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 94.64285714285714

----------Fold: 10 ----------

Gen 200: 93.17738791423002

---------- Start Test ----------

Test Accuracy: 89.28571428571429

Average Accuracy: 90.89285714285715

การทดลองที่ 3 ให้โครงสร้างของ Neural Network เป็น 30-15-5-2 และจำนวน Generation เป็น 100 จำนวนโครโมโซมเป็น 50

----------Fold: 1 ----------

Gen 100: 93.76218323586744

---------- Start Test ----------

Test Accuracy: 96.42857142857143

----------Fold: 2 ----------

Gen 100: 92.5925925925926

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 3 ----------

Gen 100: 93.17738791423002

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 4 ----------

Gen 100: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 5 ----------

Gen 100: 91.81286549707602

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 6 ----------

Gen 100: 92.20272904483431

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 7 ----------

Gen 100: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 83.92857142857143

----------Fold: 8 ----------

Gen 100: 91.61793372319688

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 9 ----------

Gen 100: 90.64327485380117

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 10 ----------

Gen 100: 92.39766081871345

---------- Start Test ----------

Test Accuracy: 89.28571428571429

Average Accuracy: 90.89285714285715

การทดลองที่ 4 ให้โครงสร้างของ Neural Network เป็น 30-15-5-2 และจำนวน Generation เป็น 200 จำนวนโครโมโซมเป็น 10

----------Fold: 1 ----------

Gen 200: 91.81286549707602

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 2 ----------

Gen 200: 91.81286549707602

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 3 ----------

Gen 200: 91.2280701754386

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 4 ----------

Gen 200: 92.98245614035088

---------- Start Test ----------

Test Accuracy: 94.64285714285714

----------Fold: 5 ----------

Gen 200: 91.42300194931774

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 6 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 89.28571428571429

----------Fold: 7 ----------

Gen 200: 91.42300194931774

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 8 ----------

Gen 200: 93.17738791423002

---------- Start Test ----------

Test Accuracy: 92.85714285714286

----------Fold: 9 ----------

Gen 200: 92.39766081871345

---------- Start Test ----------

Test Accuracy: 87.5

----------Fold: 10 ----------

Gen 200: 92.00779727095517

---------- Start Test ----------

Test Accuracy: 94.64285714285714

Average Accuracy: 91.96428571428572

วิเคราะห์ผลการทดลอง

จากการทดลองจะเห็นได้ว่า การทดลองที่ 1 การทดลองที่ 2 การทดลองที่ 3 นั้นให้ผลใกล้เคียงกันรวมถึงค่าเฉลี่ยนั้นเรียกได้ว่าเท่ากัน ไม่ว่าจะปรับโครงสร้าง จำนวน Generation แต่การทดลองที่ 4 นั้นได้ลดจำนวน Chromosome ลงจาก 50 เหลือ 10 ซึ่งเป็นการลดจำนวน Chromosome ลงจากเดิมมามากแต่กลับให้ความแม่นยำเฉลี่ยที่มากขึ้น ซึ่งแสดงให้เห็นว่าการทดลองที่ 1, การทดลองที่ 2, การทดลองที่ 3 เกิดการ over train เนื่องจากโครงสร้างของ Neural Network ถูก train ให้ fit กับข้อมูลมากเกินไป

ภาคผนวก

Code ส่วน main ของโปรแกรม main.java

```
package com.company;
import java.io.*;
import java.util.ArrayList;
import java.util.Collections;

public class Main {

    public static void main(String[] args) throws IOException {
       String path =
"C:\\Users\\nachp\\IdeaProjects\\GeneticAlgorithm\\src\\com\\company\\wdbc.txt";
        File data = new File(path);
        ArrayList<Double []> dataset = new ArrayList<Double[]>();

        BufferedReader buff = new BufferedReader(new FileReader(data));
        String read;
        while((read = buff.readLine())!= null){
            String[] line = read.split(",");
            Double[] feature = new Double[31];
            if (line[1].equals("M")){
                feature[30] = 0.0;
            }else{
                feature[30] = 1.0;
            }
            for (int i = 2; i < line.length; i++){
                feature[i-2] = Double.parseDouble(line[i]);
            }
            dataset.add(feature);
        }
        Collections.shuffle(dataset);
        System.out.println(dataset.size());
        buff.close();
        int[] model ={30,15,5,2};
        GeneticAlgorithm GA = new GeneticAlgorithm();

        for(int i = 0;i < model.length;i++){
            if(i != 0){
                System.out.print("-");
                System.out.print(model[i]);
            }
        }

        GA.initChrom(10,model);
        double acc = 0.0;
        for(int i = 0;i < 10;i++){
            System.out.println("-------------------Fold: "+(i + 1)+" ----------------
---");
            int j = (int)(i*dataset.size()*0.1);
            ArrayList<Double[]> testset = new
ArrayList<Double[]>(dataset.subList(j,(int)(j + (dataset.size()*0.1))));
            ArrayList<Double[]> trainset = (ArrayList<Double[]>) dataset.clone();
            trainset.subList(j,(int)(j+(trainset.size()*0.1))).clear();
            GA.trainNN(200,(ArrayList<Double[]>) trainset);
            System.out.println("---------------------- Start Test --------------------
---");
            acc += GA.testNN(testset);
        }
```

```
        System.out.println("Average Accuracy: " + acc/10);
    }
}
```

code ส่วน neural network NeuralNetwork.java

```
package com.company;

public class NeuralNetwork {
    double[][][] weight;
    int node;
    int[] model;

    public Double sigmoid(Double x){
        return 1.0 / (1.0 + Math.exp(-x));
    }

    public NeuralNetwork(int[] model){
        this.model = model;
        int cell = 0;
        for (int layer: model){
            if(node < layer){
                node = layer;
            }
            node += layer;
        }
        weight = new double[model.length-1][node][node];
    }

    public void initWeight(Chromosome chromosome){
        int w = 0;
        for (int i = 0; i < model.length-1;i++){
            for(int j = 0; j < model[i]; j++){
                for (int k = 0; k < model[i+1];k++){
                    weight[i][j][k] = chromosome.gene[w];
                    w++;
                }
            }
        }
    }

    public boolean feedForward(Double[] input){
        Double[][] output = new Double[model.length][node];
        output[0] = input;

        for (int i = 1; i < model.length; i++){
            for(int j = 0;j < model[i]; j++){
                Double  x = 0.0;
                for(int k = 0;k < model[i - 1];k++){
                    x += output[i - 1][k] * weight[i - 1][k][j];
                }
                output[i][j] = Math.tanh(x);
            }
        }

        if(input [30] == 0.0){
```

```
            if (output[model.length-1][0] > output[model.length-1][1]){
                return  true;
            }else return false;
        }else {
            if (output[model.length - 1][0] < output[model.length - 1][1]) {
                return true;
            } else return false;
        }
    }

}
```

code ส่วน Chromosome Chromosome.java

```
package com.company;

public class Chromosome {
    public Double[] gene;
    public Double fitness = 0.0;

    public Chromosome(Double[] gene){
        this.gene = gene;
    }
}
```

code ส่วน GeneticAlgorthim GeneticAlgorithm.java

```
package com.company;

import java.util.*;

import static jdk.nashorn.internal.objects.NativeMath.round;

public class GeneticAlgorithm {
    int[] model;
    double min = -1.0;
    double max = 1.0;
    ArrayList<Chromosome> initChrom = new ArrayList<Chromosome>();
    ArrayList<Chromosome> chroms = new ArrayList<Chromosome>();
    ArrayList<Double[]> dataSet;

    public void calFitness(Chromosome chrom){
        chrom.fitness = 0.0;
        NeuralNetwork NN = new NeuralNetwork(model);
        NN.initWeight(chrom);
        for(Double[] data: dataSet){
            if(NN.feedForward(data)){
                chrom.fitness += 1;
            }
        }
    }
```

```java
public void  findFitness(ArrayList<Chromosome> chroms){
    for (Chromosome chrom : chroms){
        calFitness(chrom);
    }
}


public ArrayList<Chromosome> random(int fit){
    int index;
    ArrayList<Chromosome> sel = new ArrayList<Chromosome>();
    for(int i = 0; i < fit; i++){
        index = new Random().nextInt(chroms.size());
        sel.add(chroms.get(index));
    }
    return  sel;
}

public ArrayList<Chromosome> crossOver(ArrayList<Chromosome> sel,int fit){
    ArrayList<Chromosome> crossover = new ArrayList<Chromosome>();
    for(int i = 0;i < fit; i++){
        int indexFather = new Random().nextInt(sel.size());
        int indexMother = new Random().nextInt(sel.size());
        Chromosome Father = sel.get(indexFather);
        Chromosome Mother = sel.get(indexMother);
        int selectLen = sel.get(0).gene.length;
        Double[] gene = new Double[selectLen];
        for (int j = 0; j < selectLen / 2; j++){
            gene[j] = Father.gene[j];
        }
        for (int k = selectLen / 2; k < selectLen; k++){
            gene[k] = Mother.gene[k];
        }
        crossover.add(new Chromosome(gene));
    }
    findFitness(crossover);
    return crossover;
}

public ArrayList<Chromosome> mutation(int fit, ArrayList<Chromosome> chromset){
    ArrayList<Chromosome> mutate = new ArrayList<Chromosome>();
    Set<Integer> index = new LinkedHashSet<Integer>();
    Random rand = new Random();
    int g = 0;
    while (index.size() < fit ){
        Integer gen = rand.nextInt(chromset.size());
        index.add(gen);
    }
    for (int i : index){
        Double[] gene = chromset.get(i).gene.clone();
        int prob = (int) (chromset.get(i).gene.length * 0.3);
        for (int j = 0;j < prob;j++){
            int k = rand.nextInt(chromset.get(i).gene.length);
            gene[k] += min + (max - min)*new Random().nextDouble();
            if (gene[k] < min){
                gene[k] = min;
            }
            if (gene[k] > max){
                gene[k] = max;
            }
        }
        mutate.add(new Chromosome(gene));
        calFitness(mutate.get(g++));
    }
```

```
        return mutate;
    }

    public void initChrom(int fit,int[] model){
        this.model = model;
        int len = 0;
        for(int i = 1;i < model.length;i++){
            len += model[i-1]*model[i];
        }
        for(int j = 0;j < fit; j++){
            Double[] gene = new Double[len];
            for(int k = 0;k < gene.length;k++){
                gene[k] = min + (max - min)*new Random().nextDouble();
            }
            initChrom.add(new Chromosome(gene));
        }
    }

    public void trainNN (int maxGen, ArrayList<Double[]> dataSet){
        this.dataSet = dataSet;
        chroms = (ArrayList<Chromosome>) initChrom.clone();
        findFitness(chroms);
        for(int i = 0;i <  maxGen;i++){
            ArrayList<Chromosome> sel = random((int)(chroms.size()*0.3));
            ArrayList<Chromosome> crossover = crossOver(sel,(int) (sel.size()*0.3));
            ArrayList<Chromosome> chromset = (ArrayList<Chromosome>) chroms.clone();
            chromset.addAll(crossover);
            int mutateSize = (int) (chroms.size()*0.3);
            ArrayList<Chromosome> mutate = mutation(mutateSize,chromset);
            Collections.sort(chromset, (o1, o2) -> (int)(o2.fitness - o1.fitness));
            chroms = new ArrayList<Chromosome>(chromset.subList(0,chroms.size() -
mutateSize));
            chroms.addAll(mutate);

        }
        System.out.println("Gen " + maxGen + ": " +
(chroms.get(0).fitness*100/dataSet.size()));
        Collections.sort(chroms, (o1, o2) -> (int)(o2.fitness - o1.fitness));
    }

    public double testNN(ArrayList<Double[]> test){
        NeuralNetwork NN = new NeuralNetwork(model);
        Chromosome bchrom = chroms.get(0);
        NN.initWeight(bchrom);
        bchrom.fitness = 0.0;
        for(Double[] data: test){
            if(NN.feedForward(data)){
                bchrom.fitness += 1;
            }
        }
        System.out.println("Test Accuracy: "  + (bchrom.fitness * 100 / test.size()));
        return (bchrom.fitness*100/test.size());
    }
}
```