Zequn Yu
CSE 151

PA 2

Q1.

If Feature 5 <= 0.5:

----------→Yes: If Feature 1 <= 420000.0:

| ----------→Yes: If Feature 17 <= 2506.0:

| | ----------→Yes: If Feature 2 <= 1.0:

.

.

.

----------→No: If Feature 5 <= 1.0:

----------→Yes: If Feature 6 <= 2.0:

| ----------→Yes: If Feature 10 <= 2.0:

Q2.

Training and test error is: 42%.

The errors I detect is to compare the predict label and true label to decide if it is an error.

Q3.

After 1 round: 4%

After 2 round: 7%

Q4.

Age is the most salient and prominent feature.

```
/*
 * Author: ZEQUN YU
 * PID: A14712777
 */

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.ListIterator;
import java.util.PriorityQueue;

public class ID3DT {

   // set variables
   static LinkedList<double[]> trainList;
   static LinkedList<double[]> testList;
   static Node root;
   static String tabs;
   public static void main(String[] args) throws IOException {

      // open file and read data from pa2train.
      File trainFile = new File("/Users/Jason/Desktop/ID3/src/pa2train.txt");
      BufferedReader rtrain = new BufferedReader(new FileReader(trainFile));
      // call function to collect data from file
      trainList = collectData(rtrain);
      //System.out.print("Read from train file is: " + trainList);
```

```
    // set a root for tree
    root = new Node( -1, -1.0, -1.0, null, null);
    /*
    System.out.println("Now the list is: " );
    for( int i = 0; i < trainList.size(); i++ ){
        for( int j = 0; j < trainList.get(i).length; j++){
            System.out.print(trainList.get(i)[j] + " ");
        }
        System.out.println(" ");
    }
    */
    // call function to build tree based on our list and root
    tabs =" ";
    //System.out.println("Start building tree");
    root = buildTree(trainList, root);
    //System.out.println("Build done");

    File testFile = new File("/Users/Jason/Desktop/ID3/src/pa2test.txt");
    BufferedReader rtest = new BufferedReader(new FileReader(testFile));
    testList = collectData(rtest);
    /*
    for( int i = 0; i < testList.size(); i++ ){
        for(int j = 0; j < testList.get(i).length; j++){
            System.out.print(testList.get(i)[j] + " ");
        }
        System.out.println(" ");
    }
    */
    int errorCount = 0;
    for (int i = 0; i < testList.size(); i++)
    {
        double[] currentData = testList.get(i);
        double predict = Stats(root, currentData);
        if (predict != currentData[4])
        {
            errorCount++;
        }

    }

    System.out.println("Dataset Size: " + testList.size());
    System.out.println("Error Count: " + errorCount);
    System.out.println("Percent Error: " + 100*(errorCount/((float)testList.size())) + "%");

}


private static double Stats(Node node, double[] dataLine) {
    if (node.lchild == null && node.rchild == null) {
        return node.feature;
    }

    if (dataLine[node.feature] <= node.threshold) {
        return Stats(node.lchild, dataLine);
    } else {
        return Stats(node.rchild, dataLine);
    }

}

// function to build tree
public static Node buildTree( LinkedList<double[]> list, Node root ){

    // check if all node has same label (pure already)
    //int label_idx = 22;

    /*
```

```
System.out.println("alist is: ");
for( int i = 0; i < alist.length; i ++ ){
    System.out.print(alist[i]);
}
*/
// use for loop to check if all label same
//System.out.println("In buildTree function");
boolean pure = true;
int label_idx = 22;
double[] alist = list.get(0);

for(int i = 1; i < list.size(); i++) {
    double[] reflist = list.get(i);
    if (alist[label  idx] != reflist[label  idx]) {
        pure = false;
    }
    alist = reflist;
}

// if result
//If it's pure, return
//System.out.println("Pure is: " + pure);
if (pure == true)
{
    System.out.println("Predict " + (int)alist[label_idx]);
    root.setLabel(alist[label  idx]);
    root.setFeature((int)alist[label_idx]);
    return root;
}

/*
System.out.println("Now the list is: " );
for( int i = 0; i < list.size(); i++ ){
    for( int j = 0; j < list.get(i).length; j++){
        System.out.print(list.get(i)[j] + " ");
    }
    System.out.println(" ");
}
*/
// Sort each feature ( 0 - 21 : 22) to get new list
//System.out.println("sort for 0");
LinkedList<double[]> f0 = sort_list(list, 0);
/*
System.out.println("After sorting f0, the list is: ");
for( int i = 0; i < f0.size(); i++ ){
    for( int j = 0; j < f0.get(i).length; j++){
        System.out.print(f0.get(i)[j] + " ");
    }
    System.out.println(" ");
}
*/

//System.out.println("sort for 1");
LinkedList<double[]> f1 = sort_list(list, 1);

/*
System.out.println("After sorting f1, the list is: ");
for( int i = 0; i < f1.size(); i++ ){
    for( int j = 0; j < f1.get(i).length; j++){
        System.out.print(f1.get(i)[j] + " ");
    }
    System.out.println(" ");
}
*/
//System.out.println("sort for 2");
LinkedList<double[]> f2 = sort_list(list, 2);
LinkedList<double[]> f3 = sort_list(list, 3);
```

```
        LinkedList<double[]> f4 = sort_list(list, 4);
        LinkedList<double[]> f5 = sort_list(list, 5);
        LinkedList<double[]> f6 = sort_list(list, 6);
        LinkedList<double[]> f7 = sort_list(list, 7);
        LinkedList<double[]> f8 = sort_list(list, 8);
        LinkedList<double[]> f9 = sort_list(list, 9);
        LinkedList<double[]> f10 = sort_list(list, 10);
        LinkedList<double[]> f11 = sort_list(list, 11);
        LinkedList<double[]> f12 = sort_list(list, 12);
        LinkedList<double[]> f13 = sort_list(list, 13);
        LinkedList<double[]> f14 = sort_list(list, 14);
        LinkedList<double[]> f15 = sort_list(list, 15);
        LinkedList<double[]> f16 = sort_list(list, 16);
        LinkedList<double[]> f17 = sort_list(list, 17);
        LinkedList<double[]> f18 = sort_list(list, 18);
        LinkedList<double[]> f19 = sort_list(list, 19);
        LinkedList<double[]> f20 = sort_list(list, 20);
        LinkedList<double[]> f21 = sort_list(list, 21);

       /*
        System.out.println("After sorting f6, the list is: ");
        for( int i = 0; i < f6.size(); i++ ){
            for( int j = 0; j < f6.get(i).length; j++){
                System.out.print(f6.get(i)[j] + " ");
            }
            System.out.println(" ");
        }
        */

        // calculate the possible threshold list and decide the threshold
        // IG(Z) = H(X) - H(X|Z)
        // get threshold and place
        //System.out.println("Start threshold");
        double[] igThres0 = target_ig(f0, 0);
        /*
        System.out.println("IG: " + igThres0[0]);
        System.out.println("Place: " + igThres0[1]);
        */
        double[] igThres1 = target_ig(f1, 1);
        //System.out.println("IG1: " + igThres0[0]);
        //System.out.println("Place1: " + igThres0[1]);
        double[] igThres2 = target_ig(f2, 2);
        double[] igThres3 = target_ig(f3, 3);
        //System.out.println("IG3: " + igThres0[0]);
        //System.out.println("Place3: " + igThres0[1]);
        double[] igThres4 = target_ig(f4, 4);
        double[] igThres5 = target_ig(f5, 5);
        double[] igThres6 = target_ig(f6, 6);
        double[] igThres7 = target_ig(f7, 7);
        double[] igThres8 = target_ig(f8, 8);
        double[] igThres9 = target_ig(f9, 9);
        double[] igThres10 = target_ig(f10, 10);
        double[] igThres11 = target_ig(f11, 11);
        double[] igThres12 = target_ig(f12, 12);
        double[] igThres13 = target_ig(f13, 13);
        double[] igThres14 = target_ig(f14, 14);
        double[] igThres15 = target_ig(f15, 15);
        double[] igThres16 = target_ig(f16, 16);
        double[] igThres17 = target_ig(f17, 17);
        double[] igThres18 = target_ig(f18, 18);
        double[] igThres19 = target_ig(f19, 19);
        double[] igThres20 = target_ig(f20, 20);
        double[] igThres21 = target_ig(f21, 21);

        // compare each threshold and decide which one will be used first
        LinkedList<double[]> threshlist = f0;
        double threshold = igThres0[1];
```

```
int axis = 0;
//System.out.println("ig0: " + igThres0[0]);
double IG = igThres0[0];


if (IG < igThres1[0])
{
   //System.out.println("in 1");
   axis = 1;
   IG = igThres1[0];
   threshlist = f1;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres1[1];
}

if (IG < igThres2[0])
{
   //System.out.println("in 2");
   axis = 2;
   IG = igThres2[0];
   threshlist = f2;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres2[1];
}

if (IG < igThres3[0])
{
   //System.out.println("in 3");
   axis = 3;
   IG = igThres3[0];
   threshlist = f3;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres3[1];
}


if (IG < igThres4[0])
{
   //System.out.println("in 4");
   axis = 4;
   IG = igThres4[0];
   threshlist = f4;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres4[1];
}

if (IG < igThres5[0])
{
   //System.out.println("in 5");
   axis = 5;
   IG = igThres5[0];
   threshlist = f5;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres5[1];
}

if (IG < igThres6[0])
{
   //System.out.println("in 6");
   axis = 6;
   IG = igThres6[0];
   threshlist = f6;
   //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
   threshold = igThres6[1];
}

if (IG < igThres7[0])
```

```
{
    //System.out.println("in 7");
    axis = 7;
    IG = igThres7[0];
    threshlist = f7;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres7[1];
}

if (IG < igThres8[0])
{
    //System.out.println("in 8");
    axis = 8;
    IG = igThres8[0];
    threshlist = f8;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres8[1];
}

if (IG < igThres9[0])
{
    //System.out.println("in 9");
    axis = 9;
    IG = igThres9[0];
    threshlist = f9;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres9[1];
}

if (IG < igThres10[0])
{
    //System.out.println("in 10");
    axis = 10;
    IG = igThres10[0];
    threshlist = f10;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres10[1];
}

if (IG < igThres11[0])
{
    //System.out.println("in 11");
    axis = 11;
    IG = igThres11[0];
    threshlist = f11;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres11[1];
}

if (IG < igThres12[0])
{
    //System.out.println("in 12");
    axis = 12;
    IG = igThres12[0];
    threshlist = f12;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres12[1];
}

if (IG < igThres13[0])
{
    //System.out.println("in 13");
    axis = 13;
    IG = igThres13[0];
    threshlist = f13;
    //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
    threshold = igThres13[1];
```

```
    }

    if (IG < igThres14[0])
    {
        //System.out.println("in 14");
        axis = 14;
        IG = igThres14[0];
        threshlist = f14;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres14[1];
    }

    if (IG < igThres15[0])
    {
        //System.out.println("in 15");
        axis = 15;
        IG = igThres15[0];
        threshlist = f15;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres15[1];
    }

    if (IG < igThres16[0])
    {
        //System.out.println("in 16");
        axis = 16;
        IG = igThres16[0];
        threshlist = f16;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres16[1];
    }

    if (IG < igThres17[0])
    {
        //System.out.println("in 17");
        axis = 17;
        IG = igThres17[0];
        threshlist = f17;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres17[1];
    }
    if (IG < igThres18[0])
    {
        //System.out.println("in 18");
        axis = 18;
        IG = igThres18[0];
        threshlist = f18;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres18[1];
    }
    if (IG < igThres19[0])
    {
        //System.out.println("in 19");
        axis = 19;
        IG = igThres19[0];
        threshlist = f19;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres19[1];
    }
    if (IG < igThres20[0])
    {
        //System.out.println("in 20");
        axis = 20;
        IG = igThres20[0];
        threshlist = f20;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres20[1];
```

```
    }
    if (IG < igThres21[0])
    {
        //System.out.println("in 21");
        axis = 21;
        IG = igThres21[0];
        threshlist = f21;
        //System.out.println("Now threshold is: " + threshlist.get((int)threshold)[axis]);
        threshold = igThres21[1];
    }

    // draw
    LinkedList left_half = new LinkedList(threshlist.subList(0, (int)threshold+1));
    LinkedList right  half = new LinkedList(threshlist.subList((int)threshold+1, threshlist.size()));

    //5. Load the threshold and axis into the node we were passed as an argument
    root.threshold = threshlist.get((int)threshold)[axis];
    root.feature = axis;

    //6. Create the two child nodes,
    root.lchild = new Node(-1, -1.0, -1.0, null, null);
    root.rchild = new Node(-1, -1.0, -1.0, null, null);

    //7. make the recursive calls, interlacing them with the prints
    System.out.println("If Feature " + (root.feature+1) + " <= " + root.threshold + ":");
    tabs = tabs + "\t";
    System.out.print(tabs + "------> Yes: ");
    tabs = tabs + "|";
    root.lchild = buildTree(left_half, root.lchild); //one priority queue
    if (tabs.length() > 1) tabs = tabs.substring(0, tabs.length()-1);
    System.out.print(tabs + "------>  No: ");
    root.rchild = buildTree(right_half, root.rchild);//one priority queue
    if (tabs.length() > 1) tabs = tabs.substring(0, tabs.length()-1);
    //8. Return the node we were passed.
    return root;
}

// function to calculate threshold
private static double[] target_ig(LinkedList<double[]> dataList, int feature)
{
    //System.out.println("In threshold function");
    //System.out.println("The passed in list is: ");
    /*
    for(int i = 0; i < dataRange.size(); i++){
        for(int j = 0; j < dataRange.get(i).length; j++){
            System.out.print(dataRange.get(i)[j] + " ");
        }
        System.out.println(" ");
    }
    */
    // set variables for threshold
    double threshold = 0.0;
    double info  gain = -1.0;
    int idx = 0;

    // use loop to calculate the H
    for (int i = 0; i < dataList.size() - 2; i++)
    {
        int num0left = 0;
        int num1left = 0;

        int j;
        for ( j = 0; j <= i; j++)
        {
            int curr_label = (int)dataList.get(j)[22];
            if (curr_label == 0)
            {
```

```
            num0left++;
        }
        else
        {
            num1left++;
        }

    }

    int num0right = 0;
    int num1right = 0;


    for (; j < dataList.size() - 1; j++)
    {
        int curr_label = (int)dataList.get(j)[22];
        if (curr_label == 0)
        {
            num0right++;
        }
        else
        {
            num1right++;
        }

    }

    //calculate conditional entropy
    double PL = (num0left + num1left ) / (double)(dataList.size());
    double PR = (num0right + num1right ) / (double)(dataList.size());

    double P0L = (num0left / (double)(dataList.size())) / PL;
    double P1L = (num1left / (double)(dataList.size())) / PL;


    double P0R = (num0right / (double)(dataList.size())) / PR;
    double P1R = (num1right / (double)(dataList.size())) / PR;


    double log0, log1;
    if (P0L == 0)
        log0 = 0;
    else
        log0 = Math.log(P0L);
    if (P1L == 0)
        log1 = 0;
    else
        log1 = Math.log(P1L);

    double entropyL = -(P0L * log0) - (P1L * log1);

    if (P0R == 0)
        log0 = 0;
    else
        log0 = Math.log(P0R);
    if (P1R == 0)
        log1 = 0;
    else
        log1 = Math.log(P1R);

    double entropyR = -(P0R * log0) - (P1R * log1);

    double condEntro = (PL * entropyL) + (PR * entropyR);

    //get normal entropy
    double pr0 = (num0left + num0right) / (double)(dataList.size());
    double pr1 = (num1left + num1right) / (double)(dataList.size());
```

```
        if (pr0 == 0)
            log0 = 0;
        else
            log0 = Math.log(pr0);
        if (pr1 == 0)
            log1 = 0;
        else
            log1 = Math.log(pr1);

        double origEntro = -(pr0 * log0) - (pr1 * log1);

        //get temporary information gain
        double tempInfoGain = origEntro - condEntro;

        //System.out.println("tempInfoGain: " + tempInfoGain);
        //System.out.println("  dataList.size(): " + dataList.size() + ", prR: " + prR);
        // check if need to update
        if (tempInfoGain > info  gain)
        {
            info_gain = tempInfoGain;
            threshold = dataList.get(i)[feature];
            idx = i;
        }
    }


    return (new double[]{(double) (info_gain), (double) (idx)});

}

// function to sort list as index of feature
public static LinkedList<double[]> sort_list( LinkedList<double[]> list, int index ){

    // at the ending of recursion, end
    if (list.size() <= 1)
        return list;

    // get the middle of index
    //System.out.println("In sort_list");
    //System.out.println("The size is: " + list.size());
    int mid_idx = list.size() / 2;
    //System.out.println("The middle index is: " + mid_idx);
    double[] middle = list.get(mid_idx);
    /*
    for( int i = 0; i < middle.length; i++ ){
        System.out.print(middle[i] + " ");
    }
    System.out.println(" ");
    */
    // add back to list later
    list.remove(mid  idx);

    // set two list to split list
    LinkedList<double[]> left = new LinkedList();
    LinkedList<double[]> right = new LinkedList();

    // for loop to sort list
    for(int i = 0; i < list.size(); i++)
    {
        // case add to left
        if (list.get(i)[index] <= middle[index])
        {
            left.add(list.get(i));
        }
        // case add to right
```

```
            else {
                right.add(list.get(i));
            }

        }

        // recursion to sort each half again
        left = sort_list(left, index);
        right = sort_list(right, index);

        // add middle back
        left.add(middle);

        // for loop to add right to left
        for(int j = 0; j < right.size(); j++)
        {
            left.add(right.get(j));
        }

        return left;
    }

    // class for root
    public static class Node{

        // set variables
        private int feature;
        private double threshold;
        private double label;
        private Node lchild;
        private Node rchild;

        public Node(){

        }

        public Node( int feature, double threshold, double label, Node lchild, Node rchild){
            this.feature = feature;
            this.threshold = threshold;
            this.label = label;
            this.lchild = lchild;
            this.rchild = rchild;

        }

        public double getFeature() {
            return feature;
        }

        public void setFeature(int feature) {
            this.feature = feature;
        }

        public double getThreshold() {
            return threshold;
        }

        public void setThreshold(double threshold) {
            this.threshold = threshold;
        }

        public double getLabel() {
            return label;
        }

        public void setLabel(double label) {
            this.label = label;
```

```
    }

    public Node getLchild() {
        return lchild;
    }

    public void setLchild(Node lchild) {
        this.lchild = lchild;
    }

    public Node getRchild() {
        return rchild;
    }

    public void setRchild(Node rchild) {
        this.rchild = rchild;
    }
}

// function to read data from file
public static LinkedList<double[]> collectData(BufferedReader br) throws IOException{
    // set a return value
    LinkedList<double[]> dataList = new LinkedList<double[]>();

    // read one line each time
    String st;
    while( (st = br.readLine()) != null){

        // set a array to store each number in line splited by whitespace
        String readLine[] = st.split("\\s");
        // get length of readLine
        int nof = readLine.length;
        // set a array to store each element in integer
        double[] readVec = new double[nof];
        // transfer String to integer
        for( int i = 0; i < nof; i++){
            // transfer
            readVec[i] = Double.parseDouble(readLine[i]);
        }
        // this line is done, collect all integer data
        dataList.add(readVec);

    }
    // return dataList
    return dataList;
}// collectData done

}
```