**Lab 7.**

Read the data from data.txt. The format of the data is four doubles followed by the class, where the class can be 1, 2, or 3. Discretize the data by removing the information after the decimal dot. For example, store 5.9 as 5. Your goal is to build a decision tree for the data. At each step, compute the information gain ratio for each of the attributes and split on the attribute that provides the highest information gain ratio. Continue each path in the decision tree until all the attributes are exhausted or change in the information gain ratio is smaller than 0.01. At the end, print the decision tree.

Expected output.

When attribute 4 has value 0

value = 2

When attribute 4 has value 1

When attribute 3 has value 3

value = 1

When attribute 3 has value 4

When attribute 1 has value 4

value = 3

When attribute 1 has value 5

value = 1

When attribute 1 has value 6

value = 1

When attribute 1 has value 7

value = 1

When attribute 3 has value 5

value = 3

When attribute 3 has value 6

value = 3

When attribute 4 has value 2

value = 3

A solution that does further splitting of nodes is also acceptable.

Design Consideration.

Create a Matrix class (about 120 lines of code). It should contain a variable that is of type a two-dimensional array. Include the following methods.

* Constructors
* private int findFrequency(int attribute, int value, ArrayList<Integer> rows) //Examines only the specified rows of the array. It returns the number of rows in which the element at position attribute (a number between 0 and 4) is equal to value.
* private HashSet<Integer> findDifferentValues(int attribute, ArrayList<Integer> rows) //Examines only the specified rows of the array. It returns a HashSet of the different values for the specified attribute.
* private ArrayList<Integer> findRows(int attribute, int value, ArrayList<Integer> rows) //Examines only the specified rows of the array. Returns an ArrayList of the rows where the value for the attribute is equal to value.
* private double log2(double number) //returns log2 of the input
* private double findEntropy(ArrayList<Integer> rows) //finds the entropy of the dataset that consists of the specified rows.
* private double findEntropy(int attribute, ArrayList<Integer> rows) //finds the entropy of the dataset that consists of the specified rows after it is partitioned on the attribute.
* private double findGain(int attribute, ArrayList<Integer> rows) // finds the information gain of partitioning on the attribute. Considers only the specified rows.
* public double computeIGR(int attribute, ArrayList<Integer> rows) // returns the Information Gain Ratio, where we only look at the data defined by the set of rows and we consider splitting on attribute.
* public int findMostCommonValue(ArrayList<Integer> rows) // returns the most common category for the dataset that is the defined by the specified rows.
* public HashMap<Integer, ArrayList<Integer>> split(int attribute, ArrayList<Integer> rows) //Splits the dataset that is defined by rows on the attribute. Each element of the HashMap that is returned contains the value for the attribute and an ArrayList of rows that have this value.

Create a Main class (about 70 lines of code). Include the following methods.

* main method
* public static int[][] process(String filename) //creates a two-dimensional array from the input file.
* public static void printDecisionTree(int[][] data, ArrayList<Integer> attributes, ArrayList<Integer> rows, int level, double currentIGR) //recursive method that prints the decision tree. It takes as input the data, the set of attributes that have not been used so far in this branch of the tree, the set of rows to examine, the current level (initially 0, use to determine how many tabs to print), and the information gain ratio from last iteration (I set it initially equal to 100, used to create terminating condition).