HOMEWORK 2

HAMSALEKHA PREMKUMAR

Instructions: Although this is a programming homework, you only need to hand in a pdf answer file. There is no need to submit the latex source or any code. You can choose any programming language, as long as you implement the algorithm from scratch (e.g. do not use Weka on questions 1 to 7).
Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file to Canvas. Please check Piazza for updates about the homework.

1 A Simplified Decision Tree

You are to implement a decision-tree learner for classification. To simplify your work, this will not be a general purpose decision tree. Instead, your program can assume that

- each item has two continuous features $\mathbf{x} \in \mathbb{R}^2$
- the class label is binary and encoded as $y \in \{0, 1\}$
- data files are in plaintext with one labeled item per line, separated by whitespace:

$$\begin{matrix} x_{11} & x_{12} & y_1 \\ & \dots & \\ x_{n1} & x_{n2} & y_n \end{matrix}$$

Your program should implement a decision tree learner according to the following guidelines:

- Candidate splits (j, c) for numeric features should use a threshold c in feature dimension j in the form of $x_{.j} \geq c$.
- c should be on values of that dimension present in the training data; i.e. the threshold is on training points, not in between training points.
- The left branch of such a split is the "then" branch, and the right branch is "else".
- Splits should be chosen using mutual information (i.e. information gain). If there is a tie you may break it arbitrarily.
- The stopping criteria (for making a node into a leaf) are that
  - the node is empty, or
  - all splits have zero mutual information
- To simplify, whenever there is no majority class in a leaf, let it predict $y = 1$.

2 Questions

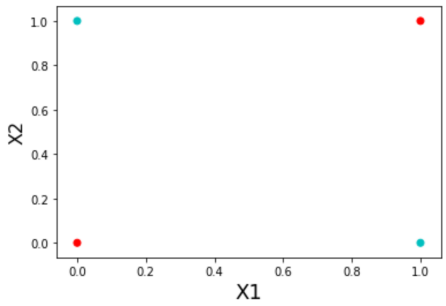
1. (Our algorithm stops at pure labels) [10 pts] If a node is not empty but contains training items with the same label, why is it guaranteed to become a leaf? Explain.

   The algorithm is designed to decide candidate split based on the feature with maximum information gain/ lowest entropy. In the case where node contains training items with same labels, information gain is 0 and no division can increase the information gain.Thus, there is no point in splitting further.

2. (Our algorithm is greedy) [10 pts] Handcraft a small training set where both classes are present but the algorithm refuses to split; instead it makes the root a leaf and stop; Importantly, if we were to manually force a split, the algorithm will happily continue splitting the data set further and produce a deeper tree with zero training error. You should (1) plot your training set, (2) explain why. Hint: you don't need more than a handful of items.

   (1).Training set that will force stop the tree from splitting is shown below

x1	x2	y
0	0	1
0	1	0
1	0	0
1	1	1



   (2).The algorithm refuses to split when the information gain is equal in all branches.It can be seen in the above data set that both sides of any horizontal/vertical decision boundary will have data points from both classes.

3. (Mutual information exercise) [10 pts] Use the training set Druns.txt. For the root node, list all candidate cuts and their mutual information. Hint: to get $\log_2(x)$ when your programming language may be using a different base, use `log(x)/log(2)`.

   Root node is x_2.

Candidate cuts	Mutual information
0.0	0.0382
1.0	0.0048
2.0	0.001
3.0	0.0163
4.0	0.0494
5.0	0.1051
6.0	0.1995
7.0	0.0382
8.0	0.1890
-1.0	0.1890
-2.0	0.0

4. (The king of interpretability) [10 pts] Decision tree is not the most accurate classifier in general. However, it persists. This is largely due to its rumored interpretability: a data scientist can easily explain a tree to a non-data scientist. Build a tree from D3leaves.txt. Then manually convert your tree to a set of logic rules. Show the tree[1] and the rules.

[1]When we say show the tree, we mean either the standard computer science tree view, or some crude plaintext representation of the tree – as long as you explain the format. When we say visualize the tree, we mean a plot in the 2D $\mathbf{x}$ space that shows how the tree will classify any points.

The tree generated from D3Leaves.txt is shown below. The notations are:

- 'cut-off' Indicates candidate cut values
- 'col' refers to the feature column name
- 'then' block is the left branch
- 'else' is the right branch
- 'val' is the label
- 'index-col' corresponds to the index of the features column in the dataset
- 'infoGain' is the mutual information
- '{}' encloses a sub-tree

```
tree from D3Leaves.txt
{'col': 'x2',
 'cutoff': 2.0,
 'else': {'val': 1},
 'index_col': 1,
 'infoGain': 0.3219280948873623,
 'then': {'col': 'x1',
          'cutoff': 10.0,
          'else': {'val': 1},
          'index_col': 0,
          'infoGain': 1.0,
          'then': {'val': 0},
          'val': 0.0},
 'val': 1.0}
```

Given one feature example,the logical rules that it would pass through for classification is given below:

```
def yDecision(x1,x2):
    if x2 < 2.0:
        if x1 < 10.0:
            return 0 #y=0
        else:
            return 1 #y=1
    else:
        return 1 #y=1
```

5. (Or is it?) [20 pts] For this question only, make sure you DO NOT VISUALIZE the data sets or plot your tree's decision boundary in the 2D $\mathbf{x}$ space. If your code does that, turn it off before proceeding. This is because you want to see your own reaction when trying to interpret a tree. You will get points no matter what your interpretation is. And we will ask you to visualize them in the next question anyway.

   - Build a decision tree on D1.txt. Show it to us in any format (e.g. could be a standard binary tree with nodes and arrows, and denote the rule at each leaf node; or Weka style plaintext tree; or as simple as plaintext output where each line represents a node with appropriate line number pointers to child nodes; whatever is convenient for you). Again, do not visualize the data set or the tree in the $\mathbf{x}$ input space. In real tasks you will not be able to visualize the whole high dimensional input space anyway, so we don't want you to "cheat" here.

```
tree from D1.txt
{'col': 'x2',
 'cutoff': 0.201829,
 'else': {'val': 1},
 'index_col': 1,
 'infoGain': 0.6690158350565576,
 'then': {'val': 0},
 'val': 1.0}
```

- Look at your tree in the above format (remember, you should not visualize the 2-D dataset or your tree's decision boundary) and try to interpret the decision boundary in human understandable English.
  The decision tree appears to be a line parallel to the $x1$ axis at $x_2 = 0.2018$. All points with $x_2 < 0.2018$ below to class 1 i.e., $y = 0$ and all points with $x_2 >= 0.2018$ belong to class 2 i.e., $y = 1$.
- Build a decision tree on D2.txt. Show it to us.

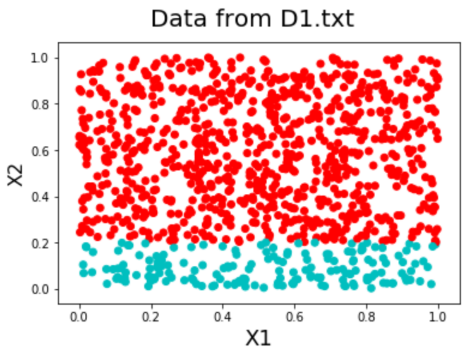

```
{'col': 'x1',
 'cutoff': 0.533076,
 'else': {'col': 'x2',
          'cutoff': 0.383738,
          'else': {'col': 'x1',
                   'cutoff': 0.550364,
                   'else': {'val': 1},
                   'index_col': 0,
                   'infoGain': 0.02904559376042053,
                   'then': {'col': 'x2',
                            'cutoff': 0.474971,
                            'else': {'val': 1},
                            'index_col': 1,
                            'infoGain': 0.5435644431995964,
                            'then': {'val': 0},
                            'val': 1.0},
                   'val': 1.0},
          'index_col': 1,
          'infoGain': 0.3583319079769226,
          'then': {'col': 'x1',
                   'cutoff': 0.761423,
                   'else': {'col': 'x2',
                            'cutoff': 0.191206,
                            'else': {'val': 1},
                            'index_col': 1,
                            'infoGain': 0.3560114223500147,
                            'then': {'col': 'x1',
                                     'cutoff': 0.90482,
                                     'else': {'col': 'x2',
                                              'cutoff': 0.037708,
                                              'else': {'col': 'x2',
                                                       'cutoff': 0.061886,
                                                       'else': {'val': 1},
                                                       'index_col': 1,
                                                       'infoGain': 0.14865258200778284,
                                                       'then': {'col': 'x2',
                                                                'cutoff': 0.053702,
                                                                'else': {'val': 0},
                                                                'index_col': 1,
                                                                'infoGain': 0.9182958340544896,
                                                                'then': {'val': 1},
                                                                'val': 1.0},
                                                       'val': 1.0},
                                              'index_col': 1,
                                              'infoGain': 0.49962073521348704,
                                              'then': {'val': 0},
                                              'val': 1.0},
                                     'index_col': 0,
                                     'infoGain': 0.40624428894524367,
                                     'then': {'col': 'x2',
                                              'cutoff': 0.169053,
                                              'else': {'col': 'x2',
                                                       'cutoff': 0.190692,
                                                       'else': {'val': 0},
                                                       'index_col': 1,
                                                       'infoGain': 0.9182958340544896,
                                                       'then': {'val': 1},
                                                       'val': 1.0},
                                              'index_col': 1,
                                              'infoGain': 0.3064509395127767,
                                              'then': {'val': 0},
                                              'val': 0.0},
                                     'val': 0.0},
                            'val': 1.0},
                   'index_col': 0,
                   'infoGain': 0.3192484516444669,
                   'then': {'col': 'x2',
                            'cutoff': 0.301105,
                            'else': {'col': 'x1',
                                     'cutoff': 0.66337,
                                     'else': {'val': 1},
                                     'index_col': 0,
                                     'infoGain': 1.0,
                                     'then': {'val': 0},
                                     'val': 0.0},
                            'index_col': 1,
                            'infoGain': 0.2689955935892812,
                            'then': {'val': 0},
                            'val': 0.0},
                   'val': 0.0},
          'val': 1.0},
 'index_col': 0,
 'infoGain': 0.22357293600240136,
 'then': {'col': 'x2',
          'cutoff': 0.639018,
          'else': {'col': 'x1',
                   'cutoff': 0.111076,
                   'else': {'col': 'x2',
                            'cutoff': 0.861,
                            'else': {'val': 1},
```

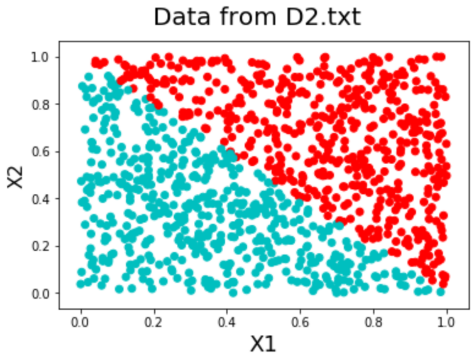
```
                        'index_col': 1,
                        'infoGain': 0.19938026829205346,
                        'then': {'col': 'x1',
                                 'cutoff': 0.33046,
                                 'else': {'val': 1},
                                 'index_col': 0,
                                 'infoGain': 0.4277709381256627,
                                 'then': {'col': 'x2',
                                          'cutoff': 0.745406,
                                          'else': {'col': 'x1',
                                                   'cutoff': 0.254049,
                                                   'else': {'val': 1},
                                                   'index_col': 0,
                                                   'infoGain': 0.5471904269481032,
                                                   'then': {'col': 'x2',
                                                            'cutoff': 0.792752,
                                                            'else': {'col': 'x1',
                                                                     'cutoff': 0.191915,
                                                                     'else': {'val': 1},
                                                                     'index_col': 0,
                                                                     'infoGain': 1.0,
                                                                     'then': {'val': 0},
                                                                     'val': 0.0},
                                                            'index_col': 1,
                                                            'infoGain': 0.3178113757536236,
                                                            'then': {'val': 0},
                                                            'val': 0.0},
                                                   'val': 1.0},
                                          'index_col': 1,
                                          'infoGain': 0.3212981208250194,
                                          'then': {'val': 0},
                                          'val': 0.0},
                                 'val': 1.0},
                        'val': 1.0},
                'index_col': 0,
                'infoGain': 0.2755449824041122,
                'then': {'col': 'x2',
                         'cutoff': 0.964767,
                         'else': {'val': 1},
                         'index_col': 1,
                         'infoGain': 0.5435644431995964,
                         'then': {'val': 0},
                         'val': 0.0},
                'val': 1.0},
        'index_col': 1,
        'infoGain': 0.3499964947105032,
        'then': {'col': 'x2',
                 'cutoff': 0.534979,
                 'else': {'col': 'x1',
                          'cutoff': 0.409972,
                          'else': {'col': 'x1',
                                   'cutoff': 0.426073,
                                   'else': {'val': 1},
                                   'index_col': 0,
                                   'infoGain': 0.2576788051033316,
                                   'then': {'col': 'x2',
                                            'cutoff': 0.597713,
                                            'else': {'val': 1},
                                            'index_col': 1,
                                            'infoGain': 1.0,
                                            'then': {'val': 0},
                                            'val': 0.0},
                                   'val': 1.0},
                          'index_col': 0,
                          'infoGain': 0.5694479740460766,
                          'then': {'val': 0},
                          'val': 0.0},
                 'index_col': 1,
                 'infoGain': 0.07574384173349029,
                 'then': {'val': 0},
                 'val': 0.0},
        'val': 0.0},
'val': 0.0}
```


- Try to interpret your D2 decision tree.
  The decision tree appears to have a staircase like decision boundary.

6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - Produce a scatter plot of the data set.





- Visualize your decision tree's decision boundary (or decision region, or some other ways to clearly visualize how your decision tree will make decisions in the feature space).
  The decision boundary in the line demarcating the two classes represented by two different colors.

Then discuss why the size of your decision trees on D1 and D2 differ. Relate this to the hypothesis space of our decision tree algorithm.
The decision rules are getting formulated so as to create a decision boundary that has a slope indicated by the scatter plot generated from D2.txt. For this multiple conditions need to be checke dat each node, hence the size of tree increases. This results in higher size in D2 than in D1 which has 0 slope.

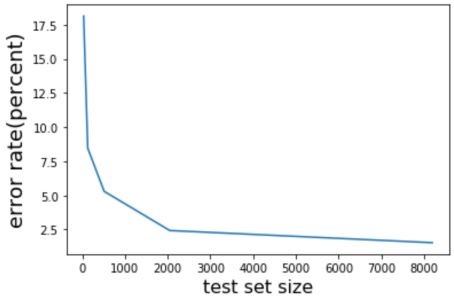
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
   - Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from the candidate training set. The subscript n in D_n denotes training set size. The easiest way is to take the first n items from the (same) permutation above. This sequence simulates the real world situation where you obtain more and more training data.
   - For each D_n above, train a decision tree. Measure its test set error err_n. Show three things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n. This is known as a learning curve (a single plot). (3) Visualize your decision trees' decision boundary (five plots). (1).

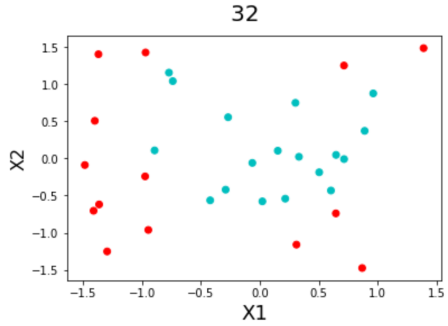
   Here, err rate=$\frac{countoffailedtestpredictions}{numberoftestexamples}$

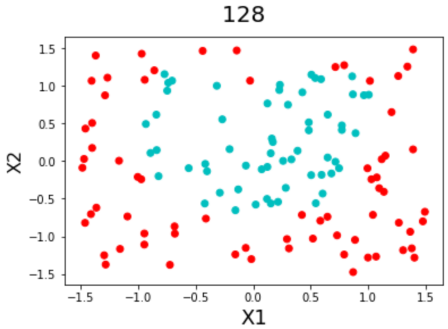
n	no. of nodes	err rate
32	3	0.18141592920353983
128	8	0.08462389380530974
512	20	0.05309734513274336
2048	51	0.024336283185840708
8192	114	0.015486725663716814

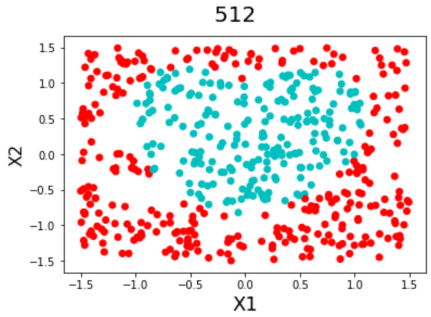
(2).

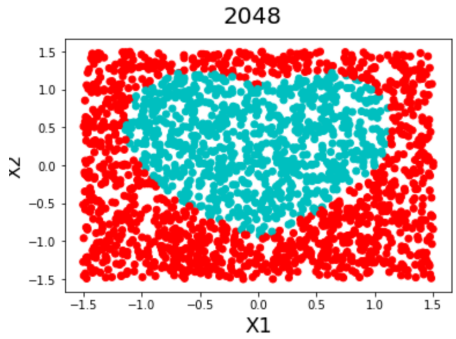


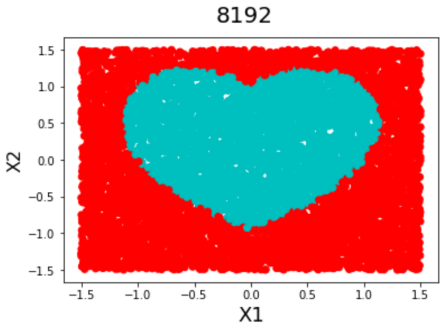
(3). Decision boundary is the boundary that demarcates the different colored data points









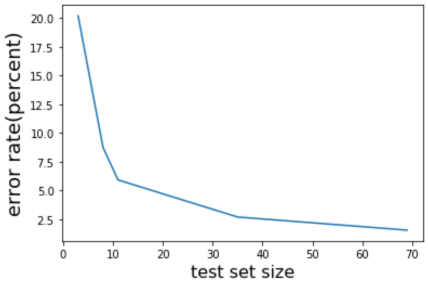


3 Weka [10 pts]

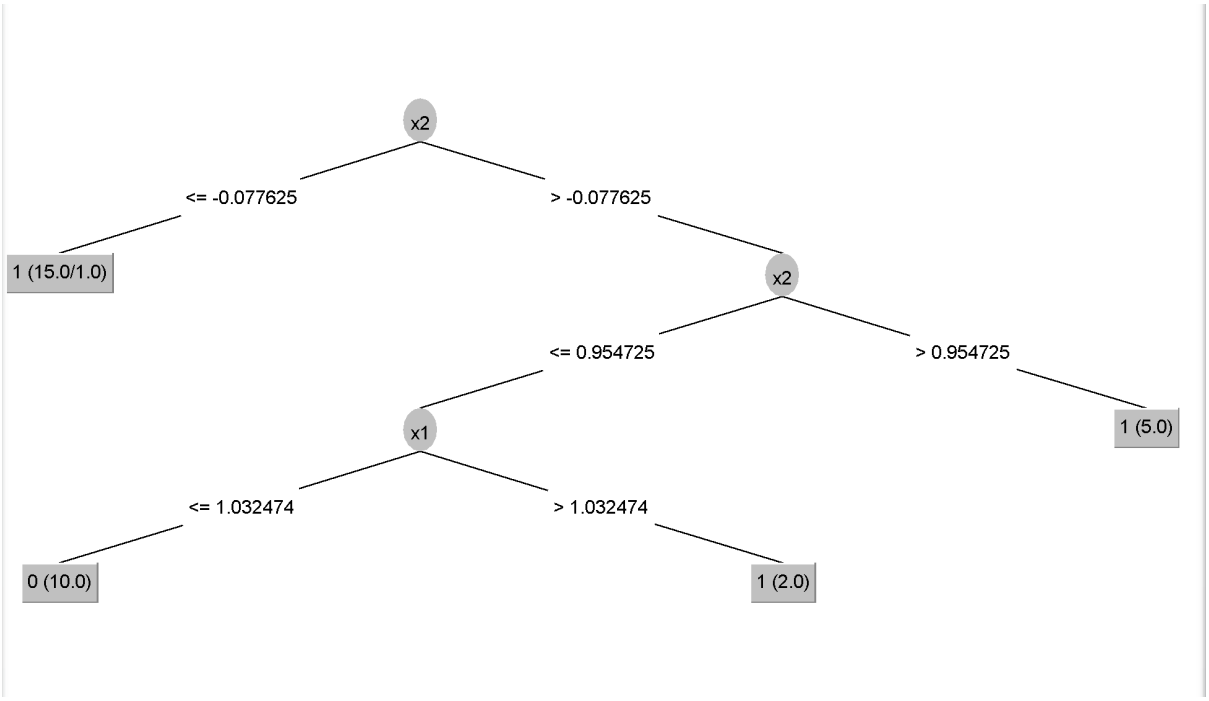
Learn to use Weka https://www.cs.waikato.ac.nz/~ml/weka/index.html. Convert appropriate data files into ARFF format. Use trees/J48 as the classifier and default settings. Produce five Weka trees for $D_{32}, D_{128} \ldots D_{8192}$. Show two things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n.
(1).

n	no. of nodes	err rate(in percent)
32	3	20.18
128	8	8.73
512	11	5.91
2048	35	2.68
8192	69	1.54

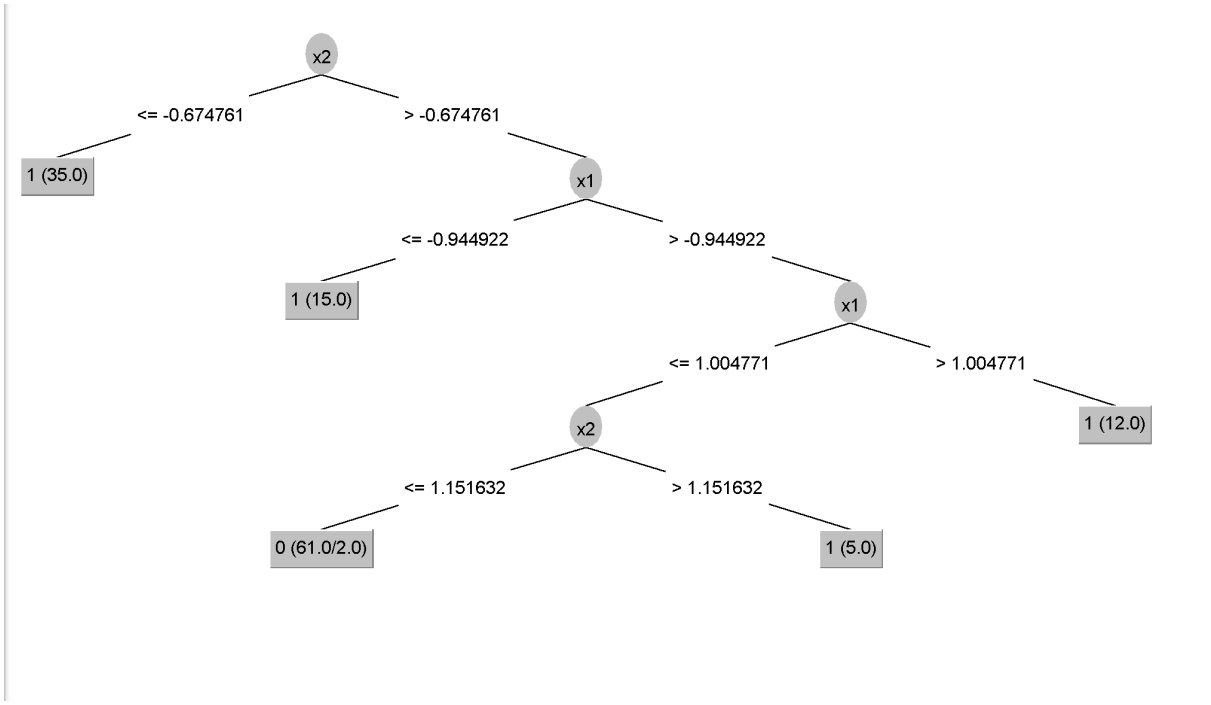
(2).



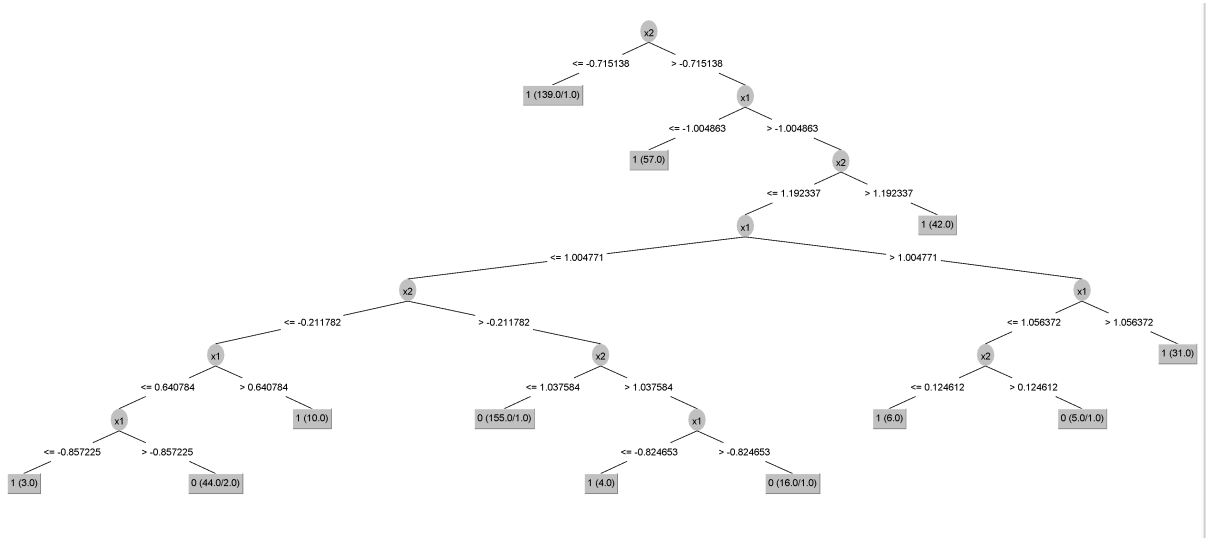
Weka trees for the 32,128,512,2048,8192 training data set
32



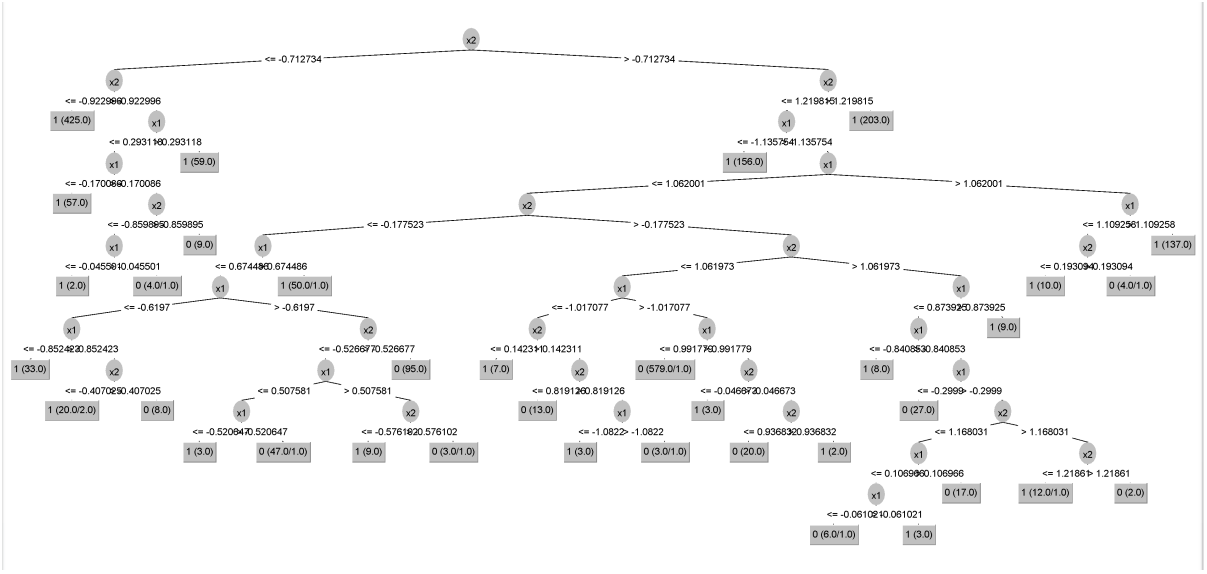
128


x2
<= -0.674761
> -0.674761
1 (35.0)
x1
<= -0.944922
> -0.944922
1 (15.0)
x1
<= 1.004771
> 1.004771
x2
1 (12.0)
<= 1.151632
> 1.151632
0 (61.0/2.0)
1 (5.0)

512


x2
<= -0.715138
> -0.715138
1 (139.0/1.0)
x1
<= -1.004863
> -1.004863
1 (57.0)
x2
<= 1.192337
> 1.192337
x1
1 (42.0)
<= 1.004771
> 1.004771
x2
x1
<= -0.211782
> -0.211782
<= 1.056372
> 1.056372
x1
x2
1 (31.0)
<= 0.640784
> 0.640784
<= 1.037584
> 1.037584
<= 0.124612
> 0.124612
x1
1 (10.0)
0 (155.0/1.0)
x1
1 (6.0)
0 (5.0/1.0)
<= -0.857225
> -0.857225
<= -0.824653
> -0.824653
1 (3.0)
0 (44.0/2.0)
1 (4.0)
0 (16.0/1.0)

2048



8192