

Homework 1 Sample Solution

1.

Iris: All attributes of “iris” are numeric, therefore ID3 of weka cannt be applied to this data set.

Contact-lenses:

```

tear-prod-rate = reduced: none
tear-prod-rate = normal
| astigmatism = no
| | age = young: soft
| | age = pre-presbyopic: soft
| | age = presbyopic
| | | spectacle-prescrip = myope: none
| | | spectacle-prescrip = hypermetrope: soft
| | astigmatism = yes
| | | spectacle-prescrip = myope: hard
| | | spectacle-prescrip = hypermetrope
| | | age = young: hard
| | | age = pre-presbyopic: none
| | | age = presbyopic: none
    
```

Cross-Validation	Training Set	2-fold	5-fold	10-fold
Error Rate	0 %	33.3333 %	16.6667 %	29.1667 %

Notes: As for details about k-fold cross-validation, you can check it out from the textbook, p111-112. Using only the training set, the test set is empty; therefore the error rate for training set is zero.

Weather-nominal:

```

outlook = sunny
| humidity = high: no
| humidity = normal: yes
outlook = overcast: yes
outlook = rainy
| windy = TRUE: no
| windy = FALSE: yes
    
```

Cross-Validation	Training Set	2-fold	5-fold	10-fold
Error Rate	0 %	71.4286 %	14.2857 %	21.4286 %

2.

- (a) If a decision learning algorithm without pruning generates a complete decision tree, one possible dataset is defined as follows:

Since the dataset has n binary-valued attributes, it has totally 2^n tuples, each of which is a possible instantiation of the n binary-valued attributes. And its corresponding class value is the decimal representation of the binary number represented by attributes. For example, if $n=5$, the generated dataset is as follows:

```
@RELATION b

@ATTRIBUTE winter    {0,1}
@ATTRIBUTE sunny     {0,1}
@ATTRIBUTE morning   {0,1}
@ATTRIBUTE cool      {0,1}
@ATTRIBUTE play      {0,1}
@ATTRIBUTE class

    {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29
, 30, 31, 32}

@DATA
0, 0, 0, 0, 0, 1
0, 0, 0, 0, 1, 2
0, 0, 0, 1, 0, 3
0, 0, 0, 1, 1, 4
0, 0, 1, 0, 0, 5
0, 0, 1, 0, 1, 6
0, 0, 1, 1, 0, 7
0, 0, 1, 1, 1, 8
0, 1, 0, 0, 0, 9
0, 1, 0, 0, 1, 10
0, 1, 0, 1, 0, 11
0, 1, 0, 1, 1, 12
0, 1, 1, 0, 0, 13
0, 1, 1, 0, 1, 14
0, 1, 1, 1, 0, 15
0, 1, 1, 1, 1, 16
1, 0, 0, 0, 0, 17
1, 0, 0, 0, 1, 18
1, 0, 0, 1, 0, 19
1, 0, 0, 1, 1, 20
```

```

1, 0, 1, 0, 0, 21
1, 0, 1, 0, 1, 22
1, 0, 1, 1, 0, 23
1, 0, 1, 1, 1, 24
1, 1, 0, 0, 0, 25
1, 1, 0, 0, 1, 26
1, 1, 0, 1, 0, 27
1, 1, 0, 1, 1, 28
1, 1, 1, 0, 0, 29
1, 1, 1, 0, 1, 30
1, 1, 1, 1, 0, 31
1, 1, 1, 1, 1, 32

```

- (b) If a decision learning algorithm without pruning generates a decision tree with only one internal node, one possible dataset is defined as follows:

Again for the dataset with n binary-valued attributes, it has totally 2^n tuples, each of which is a possible instantiation of the n binary-valued attributes. And its class variable has the opposite value of a given attribute. For example, if $n=5$, the generated dataset is as follows (Here the class variable has the opposite value of attribute “play”):

```

@RELATION a

@ATTRIBUTE winter    {0, 1}
@ATTRIBUTE sunny     {0, 1}
@ATTRIBUTE morning   {0, 1}
@ATTRIBUTE cool      {0, 1}
@ATTRIBUTE play      {0, 1}
@ATTRIBUTE class     {yes, no}

@DATA
0, 0, 0, 0, 0, yes
0, 0, 0, 0, 1, no
0, 0, 0, 1, 0, yes
0, 0, 0, 1, 1, no
0, 0, 1, 0, 0, yes
0, 0, 1, 0, 1, no
0, 0, 1, 1, 0, yes
0, 0, 1, 1, 1, no
0, 1, 0, 0, 0, yes
0, 1, 0, 0, 1, no

```

0, 1, 0, 1, 0, yes
 0, 1, 0, 1, 1, no
 0, 1, 1, 0, 0, yes
 0, 1, 1, 0, 1, no
 0, 1, 1, 1, 0, yes
 0, 1, 1, 1, 1, no
 1, 0, 0, 0, 0, yes
 1, 0, 0, 0, 1, no
 1, 0, 0, 1, 0, yes
 1, 0, 0, 1, 1, no
 1, 0, 1, 0, 0, yes
 1, 0, 1, 0, 1, no
 1, 0, 1, 1, 0, yes
 1, 0, 1, 1, 1, no
 1, 1, 0, 0, 0, yes
 1, 1, 0, 0, 1, no
 1, 1, 0, 1, 0, yes
 1, 1, 0, 1, 1, no
 1, 1, 1, 0, 0, yes
 1, 1, 1, 0, 1, no
 1, 1, 1, 1, 0, yes
 1, 1, 1, 1, 1, no

- (c) For (a), the size of the resulting tree is estimated to be $2^{n+1}-1$, which is in order of $O(2^n)$. The size of the dataset that would create such a complete decision tree on n features is 2^n . Because there are $n+1$ layers in such a complete binary tree. And for each layer i ($i=0,1,\dots,n$), there are 2^i nodes. Specifically there are 2^n leaves in the final complete binary tree. So the size of the tree, i.e. the total number of nodes of a tree, is $1+2+2^2+\dots+2^n=2^{n+1}-1$, which is of order $O(2^n)$. Also the size of the dataset corresponds to the number of leaves in the complete binary tree, which is also in order of $O(2^n)$.

For (b), the resulting tree has only three nodes, one internal node plus two leaf nodes. But the dataset still has 2^n tuples.

- (d) When $n=5$, if we use the above-defined dataset, we can get the desired decision trees as follow:

For (a):
 winter = 0
 | sunny = 0

```
| | morning = 0
| | | cool = 0
| | | | play = 0: 1
| | | | play = 1: 2
| | | cool = 1
| | | | play = 0: 3
| | | | play = 1: 4
| | morning = 1
| | | cool = 0
| | | | play = 0: 5
| | | | play = 1: 6
| | | cool = 1
| | | | play = 0: 7
| | | | play = 1: 8
| sunny = 1
| | morning = 0
| | | cool = 0
| | | | play = 0: 9
| | | | play = 1: 10
| | | cool = 1
| | | | play = 0: 11
| | | | play = 1: 12
| | morning = 1
| | | cool = 0
| | | | play = 0: 13
| | | | play = 1: 14
| | | cool = 1
| | | | play = 0: 15
| | | | play = 1: 16
winter = 1
| sunny = 0
| | morning = 0
| | | cool = 0
| | | | play = 0: 17
| | | | play = 1: 18
| | | cool = 1
| | | | play = 0: 19
| | | | play = 1: 20
| | morning = 1
| | | cool = 0
| | | | play = 0: 21
```

```

| | | | play = 1: 22
| | | | cool = 1
| | | | play = 0: 23
| | | | play = 1: 24
| sunny = 1
| | morning = 0
| | | cool = 0
| | | | play = 0: 25
| | | | play = 1: 26
| | | | cool = 1
| | | | play = 0: 27
| | | | play = 1: 28
| | morning = 1
| | | cool = 0
| | | | play = 0: 29
| | | | play = 1: 30
| | | | cool = 1
| | | | play = 0: 31
| | | | play = 1: 32

```

For (b):

play = 0: yes

play = 1: no

3. Change the code of ID3 to Gini as follows:

(1) In method `makeTree(data)`, change

a) `m_Attribute = data.attribute(Utils.maxIndex(infoGains))` to
`m_Attribute = data.attribute(Utils.minIndex(infoGains));`

b) In while loop, instead of calling method “`computeInfoGain(data, att)`”, call another method “`computeGini(data, att)`” to calculate Gini-Index values.

(2) Implement method “`computeGini(data, att)`” as follows:

```

private double computeGini(Instances data, Attribute att)
throws Exception {

double gini = 0;
Instances[] splitData = splitData(data, att);
for (int j = 0; j < att.numValues(); j++) {
    if (splitData[j].numInstances() > 0) {

```

```

gini += ((double) splitData[j].numInstances() /
        (double) data.numInstances()) *
        computeGiniIndex(splitData[j]);
    }
}
return gini;
}

```

(3) Implement method “computeGiniIndex(data)” as follows:

```

private double computeGiniIndex(Instances data) throws Exception {

    double [] classCounts = new double[data.numClasses()];
    Enumeration instEnum = data.enumerateInstances();
    while (instEnum.hasMoreElements()) {
        Instance inst = (Instance) instEnum.nextElement();
        classCounts[(int) inst.classValue()]++;
    }
    double entropy = 1;
    for (int j = 0; j < data.numClasses(); j++) {
        if (classCounts[j] > 0) {
            entropy -= Math.sqr((classCounts[j]/((double) data.numInstances()))
* (Utils.log2(classCounts[j]) - Utils.log2(data.numInstances())));
        }
    }
    return entropy ;
}

```