Practical No. 4

Roll No.: K041	Name: Anish Sudhan Nair
Class: B.Tech Cybersecurity	Batch: K2
Date of Practical: 12/02/2022	Date of Submission: 19/02/2022
Grade:	

Aim: To implement ID3 algorithm

Prerequisite:

- Working of ID3 classification algorithm
- Understanding of fundamental programming constructs in C/C++/Java
- Basic features of WEKA tool

Outcome: After successful completion of this experiment students will be able to

- Implement the process of selecting the split attribute and analyze its importance in the working of ID3 Algorithm.
- Use Classifier tab in WEKA and create a Tree based classifier model for the data set given and analyze the model created.

Theory:

The ID3 algorithm begins with the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy $H(S)$ (or information gain $IG(A)$) of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split by the selected attribute (e.g. age is less than 50, age is between 50 and 100, age is greater than 100) to produce subsets of the data. The algorithm continues to recur on each subset, considering only attributes never selected before.

Recursion on a subset may stop in one of these cases:

- Every element in the subset belongs to the same class (+ or -), then the node is turned into a leaf and labelled with the class of the examples
- There are no more attributes to be selected, but the examples still do not belong to the same class (some are + and some are -), then the node is turned into a leaf and labelled with the most common class of the examples in the subset

- There are no examples in the subset, this happens when no example in the parent set was found to be matching a specific value of the selected attribute, for example if there was no example with age >= 100. Then a leaf is created, and labelled with the most common class of the examples in the parent set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node representing the selected attribute on which the data was split, and terminal nodes representing the class label of the final subset of this branch.

A measure used from Information Theory in the ID3 algorithm and many others used in decision tree construction is that of Entropy. Informally, the entropy of a dataset can be considered to be how disordered it is. It has been shown that entropy is related to information, in the sense that the higher the entropy, or uncertainty, of some data, then the more information is required in order to completely describe that data. In building a decision tree, we aim to decrease the entropy of the dataset until we reach leaf nodes at which point the subset that we are left with is pure, or has zero entropy and represents instances all of one class (all instances have the same value for the target attribute).

We measure the entropy of a dataset,S, with respect to one attribute, in this case the target attribute, with the following calculation:

$$Entropy(S) = \sum_{i=1}^{C} p_i \log_2 p_i$$

where Pi is the proportion of instances in the dataset that take the ith value of the target attribute, which has C different values.

This probability measures give us an indication of how uncertain we are about the data. And we use a log2 measure as this represents how many bits we would need to use in order to specify what the class (value of the target attribute) is of a random instance.

We can use a measure called Information Gain, which calculates the reduction in entropy (Gain in information) that would result on splitting the data on an attribute, A.

$$Gain\ (S, A) = Entropy\ \ (S) - \sum_{v \in A} \frac{|S_v|}{|S|} Entropy\ \ (S_v)$$

where v is a value of A , |Sv| is the subset of instances of S where A takes the value v,
and |S| is the number of instances

****

(TO BE COMPLETED BY STUDENTS)

Roll No.: K041	Name: Anish Sudhan Nair
Class: B.Tech Cybersecurity	Batch: K2
Date of Practical: 12/02/2022	Date of Submission: 19/02/2022
Grade:	

a. Implement an ID3 algorithm for selecting the first splitting attribute in the Height data set given below.

Name	**Gender**	**Height**	**Output1(Correct)**
Kristina	F	1.6m	Short
Jim	M	2m	Tall
Maggie	F	1.9m	Medium
Martha	F	1.88m	Medium
Stephanie	F	1.7m	Short
Bob	M	1.85m	Medium
Kathy	F	1.6m	Short
Dave	M	1.7m	Short
Worth	M	2.2m	Tall
Steven	M	2.1m	Tall
Debbie	F	1.8m	Medium
Todd	M	1.95m	Medium
Kim	F	1.9m	Medium
Amy	F	1.8m	Medium
Wynette	F	1.75m	Medium

Code:

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

struct treeAttributes{
  char name;
  char gender;
  float height;
  char heightRange;
  char output;
} attribute[15];

int n=15;
char
name[15]={"Kristina","Jim","Maggie","Martha","Stephanie","Bob","Kathy","Dave","Worth","Steven","Debbie","Todd"
, "Kim","Amy","Wynette"};
char gender[15]={"f","m","f","f","f","m","f","m","m","m","f","m","f","f","f"};
float height[15]={1.6,2,1.9,1.88,1.7,1.85,1.6,1.7,2.2,2.1,1.8,1.95,1.9,1.8,1.75};
char output[15]={"s","t","m","m","s","m","s","s","t","t","m","m","m","m","m"};
char genderValues[2]={"m","f"};
char heightValues[6]={"1","2","3","4","5","6"};
char heightRange[15];

char heightDiscretiser(float height){

  if ((height>0)||(height<=1.6)) {
    return "1";
  }
  if ((height>1.6)||(height<=1.7)) {
    return "2";
  }
  if ((height>1.7)||(height<=1.8)) {
    return "3";
  }
  if ((height>1.8)||(height<=1.9)) {
    return "4";
  }
  if ((height>1.9)||(height<=2.0)) {
    return "5";
  }
  if (height>2.0) {
    return "6";
  }
  return "0";
}

void valueAssigner(){

  for (int i = 0; i < 15; i++) {
    attribute[i].name=name[i];
    attribute[i].gender=gender[i];
    attribute[i].height=height[i];
    attribute[i].output=output[i];
    attribute[i].heightRange=heightDiscretiser(height[i]);
    heightRange[i]=heightDiscretiser(height[i]);
  }
}

unsigned int Log2n(float n)
```

```
{
    return (n > 1) ? 1 + Log2n(n / 2) : 0;
}

float probabilityLog(int numerator, int denominator){

  float value = -(numerator/denominator)*(Log2n(numerator/denominator));

  return value;
}

int baseCounter(char match, char array[]){

  int count=0;

  for (int i = 0; i < n; i++) {
    if (match==array[i]) {
      count++;
    }
  }

  return count;
}

int counter(char match, char array[], char condition){

  int count=0;

  for (int i = 0; i < n; i++) {
    if ((match==array[i])&&(condition==output[i])) {
      count++;
    }
  }

  return count;
}

float baseEntropy(){

  int yesShort=baseCounter("s",output); //can make a function for this
  int noShort=n-yesShort;
  int yesMedium=baseCounter("m",output);
  int noMedium=n-yesMedium;
  int yesTall=baseCounter("t",output);
  int noTall=n-yesTall;

  float entropy=probabilityLog(yesShort,n)+probabilityLog(yesMedium,n)+probabilityLog(yesTall,n);

  return entropy;
}

float attributeEntropyCalculator(char attribute, char array[]){

  //make more modular

  int denominator=baseCounter(attribute,array);
  int numeratorShort=counter(attribute,array,"s");
  int numeratorMedium=counter(attribute,array,"m");
  int numeratorTall=counter(attribute,array,"t");
```

```
  float entropy=probabilityLog(numeratorShort,denominator)+probabilityLog(numeratorMedium,denominator)+probabilityLog(numeratorTall,denominator);

  return entropy;
}

float weightedSumCalculator(char array[], char values[], int  n){

  float value;

  for (int i = 0; i < n; i++) {
    value+=attributeEntropyCalculator(values[i], array);
  }

  return value;
}

float InfoGainCalculator(float classEntropy, float weightedSum){

  float value=classEntropy-weightedSum; //these values will be recorded form baseEntropy & weightedSumCalculator in main and then passed here

  return value;
}

int main() {

  valueAssigner();

  float weightedSumGender=weightedSumCalculator(gender,genderValues,2);
  float weightedSumHeight=weightedSumCalculator(height,heightValues,7);

  float classEntropy=baseEntropy();

  float InfoGainGender=InfoGainCalculator(classEntropy, weightedSumGender); //maybe create a struct for this
  float InfoGainHeight=InfoGainCalculator(classEntropy, weightedSumHeight);

  if (InfoGainGender>+InfoGainHeight) {
    printf("The first splitting attribute is Gender\n");
  }
  else {
    printf("The first splitting attribute is Height\n");
  }

  return 0;
}
```

Output:

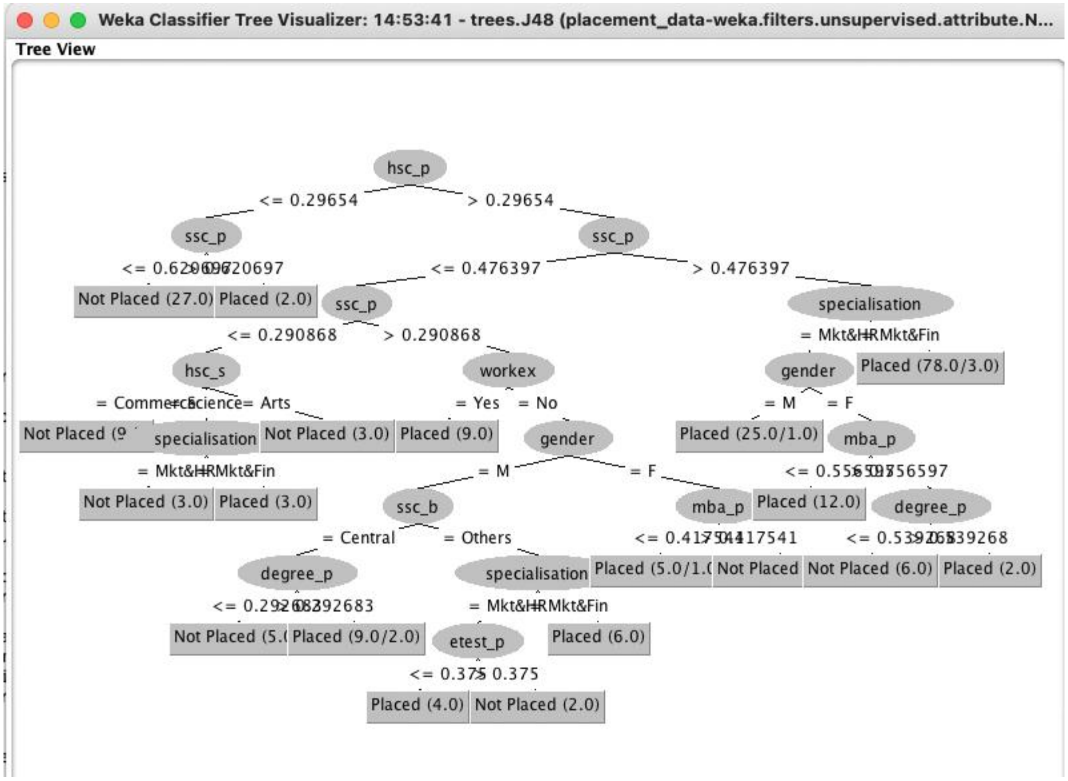
```
(base) anish@PotatoBook lab4 % ./id3
The first splitting attribute is Height
(base) anish@PotatoBook lab4 %
```

b. Using WEKA tool: For the placement data set given (Placement_Data.csv), construct a decision tree using J48 and classify the tuple,

<F,0.950526,Others,0.461285,Others,Science,0.756098,Comm&Mgmt,Yes,0.791667,Mkt&Fin,0.808471,0.081081,Placed>

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     placement_data-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
Instances:    215
Attributes:   15
              sl_no
              gender
              ssc_p
              ssc_b
              hsc_p
              hsc_b
              hsc_s
              degree_p
              degree_t
              workex
              etest_p
              specialisation
              mba_p
              salary
              status
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

hsc_p <= 0.29654
|   ssc_p <= 0.620697: Not Placed (27.0)
|   ssc_p > 0.620697: Placed (2.0)
hsc_p > 0.29654
|   ssc_p <= 0.476397
|   |   ssc_p <= 0.290868
|   |   |   hsc_s = Commerce: Not Placed (9.0)
|   |   |   hsc_s = Science
|   |   |   |   specialisation = Mkt&HR: Not Placed (3.0)
|   |   |   |   specialisation = Mkt&Fin: Placed (3.0)
|   |   |   hsc_s = Arts: Not Placed (3.0)
|   |   ssc_p > 0.290868
|   |   |   workex = Yes: Placed (9.0)
|   |   |   workex = No
|   |   |   |   gender = M
|   |   |   |   |   ssc_b = Central
|   |   |   |   |   |   degree_p <= 0.292683: Not Placed (5.0)
|   |   |   |   |   |   degree_p > 0.292683: Placed (9.0/2.0)
|   |   |   |   |   ssc_b = Others
|   |   |   |   |   |   specialisation = Mkt&HR
|   |   |   |   |   |   |   etest_p <= 0.375: Placed (4.0)
|   |   |   |   |   |   |   etest_p > 0.375: Not Placed (2.0)
|   |   |   |   |   |   specialisation = Mkt&Fin: Placed (6.0)
|   |   |   |   gender = F
|   |   |   |   |   mba_p <= 0.417541: Placed (5.0/1.0)
|   |   |   |   |   mba_p > 0.417541: Not Placed (5.0)
|   ssc_p > 0.476397
|   |   specialisation = Mkt&HR
|   |   |   gender = M: Placed (25.0/1.0)
|   |   |   gender = F
|   |   |   |   mba_p <= 0.556597: Placed (12.0)
|   |   |   |   mba_p > 0.556597
|   |   |   |   |   degree_p <= 0.539268: Not Placed (6.0)
|   |   |   |   |   degree_p > 0.539268: Placed (2.0)
|   |   specialisation = Mkt&Fin: Placed (78.0/3.0)

Number of Leaves  :     19

Size of the tree :      36


Time taken to build model: 0.13 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         178               82.7907 %
Incorrectly Classified Instances        37               17.2093 %
Kappa statistic                          0.5905
Mean absolute error                      0.1974
Root mean squared error                  0.3923
Relative absolute error                 45.9158 %
Root relative squared error             84.68   %
Total Number of Instances              215

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.892    0.313    0.863      0.892    0.877      0.591    0.843     0.891     Placed
                 0.687    0.108    0.742      0.687    0.713      0.591    0.843     0.708     Not Placed
Weighted Avg.    0.828    0.249    0.825      0.828    0.826      0.591    0.843     0.834

=== Confusion Matrix ===

   a   b   <-- classified as
 132  16 |   a = Placed
  21  46 |   b = Not Placed
```



Therefore, on travelling the branches of the decision tree for the given tuple, we eventually reach specialization for which the tuple has Mkt&Fin and so is classified as Placed.

Questions to be answered:

a. What attributes do you think might be crucial in the decision making process of classification?

➔ The decision to choose a splitting attribute depends entirely on the information gain which is again dependent on the entropy. Thus, the attributes with least entropy ot most information gain would be most influential in charting a decision tree.

b. Does training a decision tree using cross validation have any improvement on the classification accuracy? Comment.

➔ Yes, cross validation (k fold) works by training the model on subsets of the entire dataset to ensure that all underlying trends are discovered, has a good ration of testing points (k folds -> k subsets -> k points) and iterates on the same data multiple times.

Thus, by virtue of its very mechanism, cross validation yields higher levels of accuracy.

- ➔ Results from weka post cross validation training

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     placement_data-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0-weka.filters.unsupervised.attribute.Remove-R13-14-we
Instances:    215
Attributes:   11
              gender
              ssc_p
              ssc_b
              hsc_p
              hsc_b
              hsc_s
              degree_p
              degree_t
              workex
              specialisation
              status
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

J48 pruned tree
------------------

hsc_p <= 0.29654
|   ssc_p <= 0.620697: Not Placed (27.0)
|   ssc_p > 0.620697: Placed (2.0)
hsc_p > 0.29654
|   ssc_p <= 0.476397
|   |   ssc_p <= 0.290868
|   |   |   hsc_s = Commerce: Not Placed (9.0)
|   |   |   hsc_s = Science
|   |   |   |   specialisation = Mkt&HR: Not Placed (3.0)
|   |   |   |   specialisation = Mkt&Fin: Placed (3.0)
|   |   |   hsc_s = Arts: Not Placed (3.0)
|   |   ssc_p > 0.290868
|   |   |   workex = Yes: Placed (9.0)
|   |   |   workex = No
|   |   |   |   gender = M
|   |   |   |   |   ssc_b = Central
|   |   |   |   |   |   degree_p <= 0.292683: Not Placed (5.0)
|   |   |   |   |   |   degree_p > 0.292683: Placed (9.0/2.0)
|   |   |   |   |   ssc_b = Others: Placed (12.0/2.0)
|   |   |   |   gender = F
|   |   |   |   |   ssc_b = Central
|   |   |   |   |   |   degree_p <= 0.571463: Placed (5.0/1.0)
|   |   |   |   |   |   degree_p > 0.571463: Not Placed (2.0)
|   |   |   |   |   ssc_b = Others: Not Placed (3.0)
|   ssc_p > 0.476397: Placed (123.0/10.0)

Number of Leaves  :     14

Size of the tree :      26
```

```
Time taken to build model: 0.01 seconds

=== Evaluation on training set ===

Time taken to test model on training data: 0 seconds

=== Summary ===

Correctly Classified Instances         200               93.0233 %
Incorrectly Classified Instances        15                6.9767 %
Kappa statistic                          0.8268
Mean absolute error                      0.1229
Root mean squared error                  0.2479
Relative absolute error                 28.5968 %
Root relative squared error             53.5163 %
Total Number of Instances              215

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    0.224    0.908      1.000    0.952      0.839    0.917     0.932     Placed
                 0.776    0.000    1.000      0.776    0.874      0.839    0.917     0.887     Not Placed
Weighted Avg.    0.930    0.154    0.937      0.930    0.928      0.839    0.917     0.918

=== Confusion Matrix ===

   a   b   <-- classified as
 148   0 |   a = Placed
  15  52 |   b = Not Placed
```

c. How can you convert the above generated Decision tree into a series of *if - then – rules*

➔ Taking help of Weka:

Classifier output

```
=== Run information ===

Scheme:       weka.classifiers.trees.J48 -C 0.25 -M 2
Relation:     placement_data-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0
Instances:    215
Attributes:   15
              sl_no
              gender
              ssc_p
              ssc_b
              hsc_p
              hsc_b
              hsc_s
              degree_p
              degree_t
              workex
              etest_p
              specialisation
              mba_p
              salary
              status
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
------------------

hsc_p <= 0.29654
|   ssc_p <= 0.620697: Not Placed (27.0)
|   ssc_p > 0.620697: Placed (2.0)
hsc_p > 0.29654
|   ssc_p <= 0.476397
|   |   ssc_p <= 0.290868
|   |   |   hsc_s = Commerce: Not Placed (9.0)
|   |   |   hsc_s = Science
|   |   |   |   specialisation = Mkt&HR: Not Placed (3.0)
|   |   |   |   specialisation = Mkt&Fin: Placed (3.0)
|   |   |   hsc_s = Arts: Not Placed (3.0)
|   |   ssc_p > 0.290868
|   |   |   workex = Yes: Placed (9.0)
|   |   |   workex = No
|   |   |   |   gender = M
|   |   |   |   |   ssc_b = Central
|   |   |   |   |   |   degree_p <= 0.292683: Not Placed (5.0)
|   |   |   |   |   |   degree_p > 0.292683: Placed (9.0/2.0)
|   |   |   |   |   ssc_b = Others
|   |   |   |   |   |   specialisation = Mkt&HR
|   |   |   |   |   |   |   etest_p <= 0.375: Placed (4.0)
|   |   |   |   |   |   |   etest_p > 0.375: Not Placed (2.0)
|   |   |   |   |   |   specialisation = Mkt&Fin: Placed (6.0)
|   |   |   |   gender = F
|   |   |   |   |   mba_p <= 0.417541: Placed (5.0/1.0)
|   |   |   |   |   mba_p > 0.417541: Not Placed (5.0)
|   ssc_p > 0.476397
|   |   specialisation = Mkt&HR
|   |   |   gender = M: Placed (25.0/1.0)
|   |   |   gender = F
|   |   |   |   mba_p <= 0.556597: Placed (12.0)
|   |   |   |   mba_p > 0.556597
|   |   |   |   |   degree_p <= 0.539268: Not Placed (6.0)
|   |   |   |   |   degree_p > 0.539268: Placed (2.0)
|   |   specialisation = Mkt&Fin: Placed (78.0/3.0)

Number of Leaves  :     19

Size of the tree :      36


Time taken to build model: 0.13 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         178               82.7907 %
Incorrectly Classified Instances        37               17.2093 %
Kappa statistic                          0.5905
Mean absolute error                      0.1974
Root mean squared error                  0.3923
Relative absolute error                 45.9158 %
Root relative squared error             84.68   %
Total Number of Instances              215

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.892    0.313    0.863      0.892    0.877      0.591    0.843     0.891     Placed
                 0.687    0.108    0.742      0.687    0.713      0.591    0.843     0.708     Not Placed
Weighted Avg.    0.828    0.249    0.825      0.828    0.826      0.591    0.843     0.834

=== Confusion Matrix ===

   a   b   <-- classified as
 132  16 |   a = Placed
  21  46 |   b = Not Placed
```

****