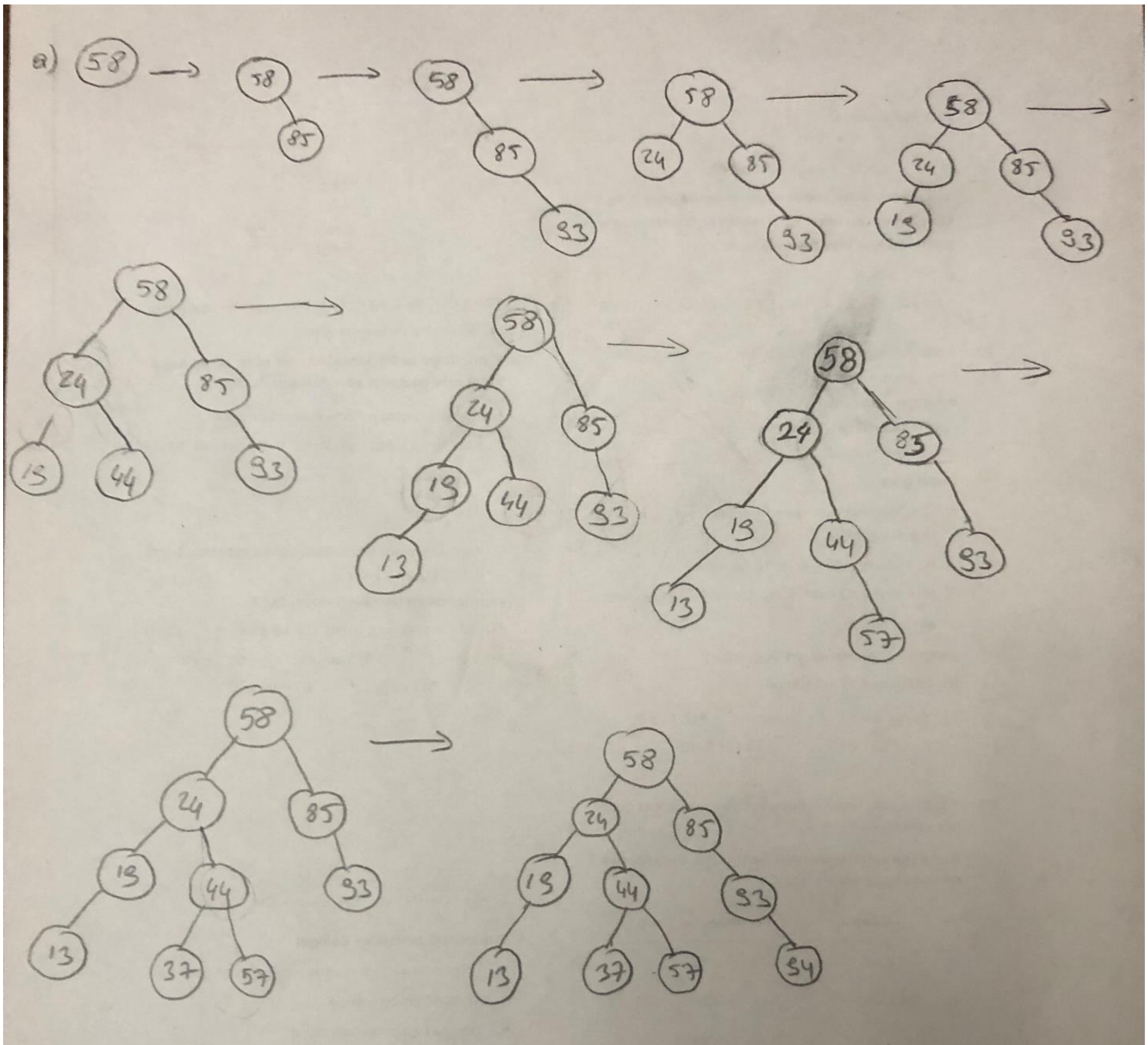
HOMEWORK 2 REPORT

Muhammed Emre YILDIZ

21702825 Sc:02

Quesiton 1’s Solutions are below

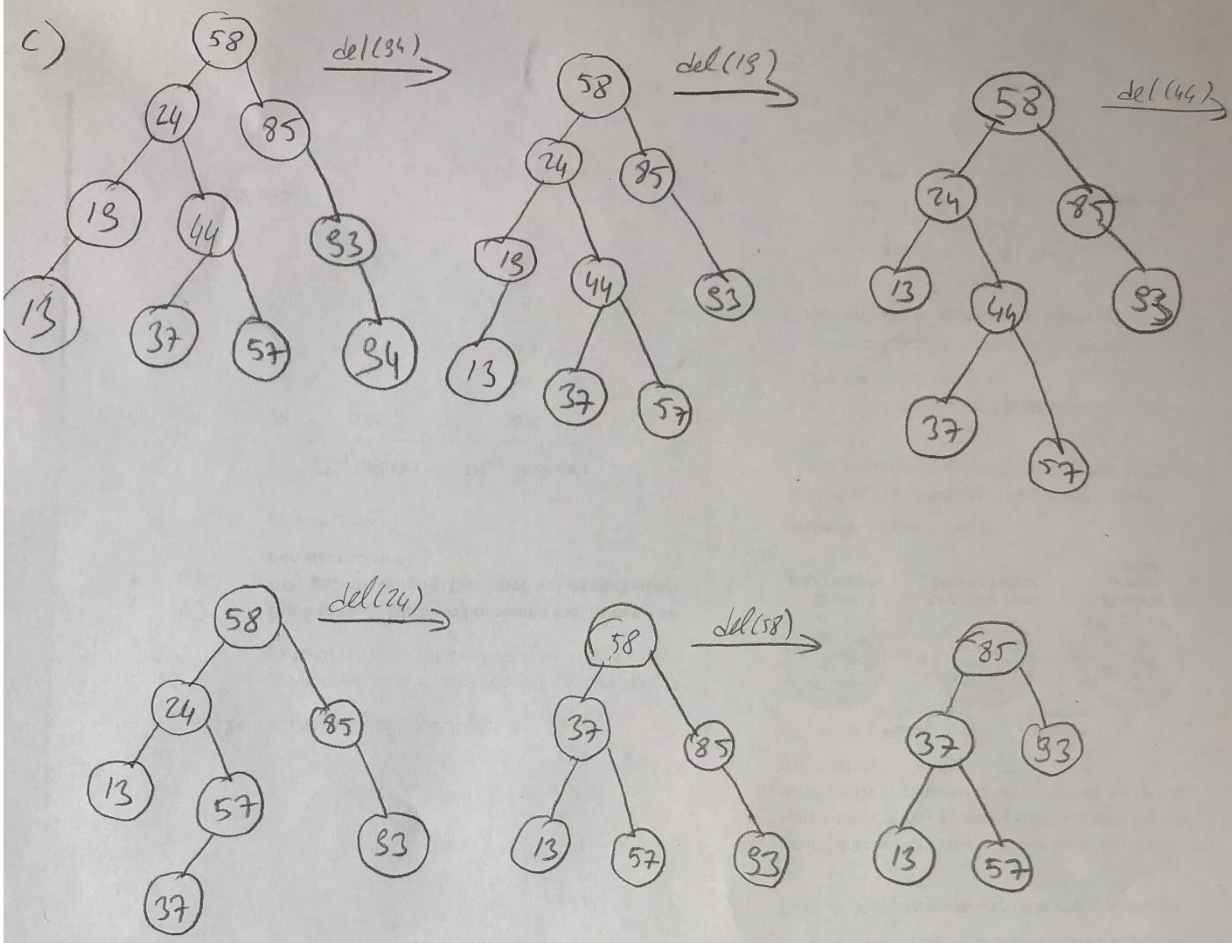


Preorder: 58, 24, 19, 13, 44, 37, 57, 85, 93, 94

Inorder: 13, 19, 24, 37, 44, 57, 58, 85, 93, 94

PostOrder: 13, 19, 37, 57, 44, 24, 94, 93, 85, 58.

c)


58
24
85
19
44
93
13
37
57
94
del(94)
58
24
85
19
44
93
13
37
57
del(19)
58
24
85
13
44
93
37
57
del(44)
58
24
85
13
57
93
37
del(24)
58
37
85
13
57
93
del(58)
85
37
93
13
57

Question 2 Implementation

- **Double calculateEntropy():**

In this function the entroy is calculated by the rules. First the sum of the class is calculated after that the calculation formula is implemented.

```
for (int i = 0; i < numClasses; i++)
{
   if (classCounts[i] != 0)
   {

   double tmp = (double)classCounts[i]/sum;
   ent +=(double)tmp*log2(tmp);
   }
}
```

Time complexity: O(n) (n is numClasses)

- **Double calculateInformationGain();**

1. In this function, firstly the number of the labels is calculated with a for loop.

```
for (int i = 0; i < numSamples; i++)

{

        if(maxL < labels[i]){

        maxL = labels[i];

        }

}
```

2. After that 3 int array is created with the size is the number of the labels (arr, arrf,arrt). First array (arr) counts the number of all labels with the same index. Second array (arrf) counts the number of each label which does not have the particular feature which is determined by featureID with the same method in first array. And the last array (arrt) is counts the number of each label which has the particular feature wtih the same method in first and second array.

```
for (int i = 0; i < numSamples; i++)
   {
         if(!usedSamples[i])
         {
                  if(data[i][featureId])
                  {
                            arrt[labels[i]-1]++;
                  }

                  else
                  {
                           arrf[labels[i]-1]++;
                  }
         }
   }
```

In addition, the probablity of the true features and false features is calculated. At the end, the entropy of the each array is calculated and returned with the formula which is given for information gain. Moreover, this function calculated respect the usedSamples. If a sample used, it is ignored.

Time Complexity: O(n) (n is number of Samples)

- ## Class DecisionTreeNode:

The purpose of the class is creating nodes which have data, right node and left node. And also there is just a **isLeaf()** method that is check a node that it is leaf or not.

Time Comlexity: O(1)

- ## Void train():

In this function two arrays are created which are usedSamples and usedFeatures and call the trainTree function with these parameters and root.

Time Complexity: O(numSamples + numFeatures + O(trainTree()))

- **Void trainTree():**

  This function is a recursion function.

1. In this function firstly the trainCalculate function is called in order to calculate that which feature has the biggest information gain respect to all FeatureIDs. Also this function uses calculateInformationGain so the information gain is calculated with respect to usedSampels as well. If the all samples are used, the function returns -1. Creating two more arrays for see the usedSampels of left and right child of the root which are usedSamplesLeft and UsedSamplesRight. UsedSamplesLeft's purpose is seeing that which samples have the particular feature. UsedSamplesRights purpose is the opposite of UsedSamplesLeft. If sample Also one more array is created which has a purpose that checking if all features are used or not.

2. There is two exit case of this function. First, if all featured are used and there is not a certain decision tree ocuurs, the error message is printed and function returns. The other exit case is if the trainCalculate function returns -1, it means that all samples are used, it can be said that rest of labels are pure so it would be the leaf of the certain tree and function returns.

3. In the recursion part of the function is, checking the particular feature which is returned by the trainCalculate of all samples, if a sample has this feature, usedSamplesRight's same index becomes true, if sample does not have, usedSamplesLeft's same index becomes false. In the same for loop, if root is null creating new decisionTreeNode as a root.

```
for (int i = 0; i < numSamples; i++)
{
   if(!usedSamples[i]){
      if(root == NULL)
      {
         root = new DecisionTreeNode(f);
         usedFeatures[f] = true;

      }

      if(data[i][f])
      {
         usedFeatures[f] = true;
         usedSamplesRight[i] = true;
      }

      else
      {
```

```
                usedFeatures[f] = true;
                usedSamplesLeft[i] = true;
                }
            }
        }
```

4. After the all steps, the function is called recursively. Firstly it is called for left tree with usedSampesLeft and root->left. Secondly it is called for the right sub tree with usedSamplesRight and root->right.

```
trainTree(data, labels, numSamples, numFeatures, usedSamplesLeft, usedFeatures,     root->left);
trainTree(data, labels, numSamples, numFeatures, usedSamplesRight, usedFeatures, root->right);
```

Time Complexity: O(log(numFeatures)*numSamples*numFeatures)

- **Void train()**

This functions purpose is reading data from the txt file and determines the parameter for upper train function.

1. Opening the file and get every lines character and store them in dataRow array.
2. After that last character of the each line represents the label of the sample so store it in labels array.
3. Creating the two dimensional data array with respcect to each dataRow
4. Closing the file.
5. Calling the upper train function with the data, labels, numSamples and numFeatures.
6. Deleting data.

Time Complexity: O(n*m + O(train())) (n: numSample, m: numFeatures)

- **Void predict()**

This function can be called after the train function called. Otherwise it just returns -1 and print error message.

The function is traversing the decision tree which is created by the train method with respect to the data’s features. When coming to a leaf, it returns the leaf’s data which represent the prediction of the class which suppose to be label of the particular data.

Time Complexity: O(log(n)) (n: the number of the features)

- **Void test()**

This function can be called after the train function is called. The purpose of the function is predict the labels of test data with respect to the decision tree.

1. Calculating every sample of data's predict and store them in a array.
2. Checking each prediction that if it is same with the given label or not.
3. Calculating the rate of true prediction /false prediction
4. Return the rate.

Time Complexity: O(log(m)*n) (n: numSamples, m: numFeatures)

- **Void test()**

This function reads the test data from the txt file. Its algorithm is the same with the train() function which is explained above. There is just a small addition, the numFeatures is not given so numFeatures is calculated by the first line of the given file. After that the test() method is called like train() method.

Time Complexity: O(log(m)*n*m) (n: numSample, m: numFeatures).

- **Void print()**

This function can be called after the train function is called. It calls print2D function. Print2D() function runs recursively. It is called for left child and the number of called function is stored so the number of "tab" is determined. After left child becomes null, they are printed with new line. The number of tab adjusted respect to the counter and right child is called. Also, if the node is leaf it printed with class message.

```
print2D(root ->left,s);

   cout << endl;

   for(int i = c; i < s; i++)
      cout << "\t";
   if(root ->isLeaf())
      cout<< "class: "<< root ->data << "\n";
   else
      cout<< root ->data << "\n";

   print2D(root ->right, s);
```

Time Complexity: Worst case: O(n*log(n)) Avarage case: O($\log^2$ (n))

(n: number of nodes).