Vivekanand Education Society's Institute Of Information Technology

Collector Colony, Kurla (East),
Mumbai – 400074, Tel: 91-22-261532532

A Project Report

On

"Decision Tree Stimulation"

Submitted By

Ninad Patil (Roll No 33)

Sarath Shankaranarayanan (Roll No 41)

(Advanced Web Technology & Data Mining and Business Intelligence Lab)

MINI PROJECT

CERTIFICATE

This Mini Project duly signed in this Documentation represents the bonafide works by:

NAME: NINAD A PATIL **ROLL NO: 33**
NAME: SARATH SHANKARANARAYANAN **ROLL NO: 41**

For SEMESTER - IV of Second Year of the Master in Computer Application (MCA) [Afternoon Shift] in the Computer Laboratory of this College during the academic year 2020- 2021

Lecturer In-Charge **Head, Department of MCA**

Lab Incharge **External Examiner**

ACKNOWLEGEMENT

It is in good fortune that we find the opportunity to express our deep sense of gratitude to all those people who helped us with their guidance and assistance without which this project would not be possible.

The successful completion of any task would be incomplete without the mention of those people whose cooperation made it possible, whose constant guidance and encouragement crown all our efforts with success.

We would like to give our heartiest thanks to the Project guide, **Prof. MANJU AHUJA AND Prof. DHANAMMA JAGLI** giving us their precious time, incessant encouragement, and for their valuable advice and guidance without which this project would not have seen the light of day.

We also thank our group members and colleagues who have helped in the successful completion of this project. And last but not least, we would like to thank all those who contributed to this project either morally or materially. Thank you all.

INDEX

INTRODUCTION

A decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponds to records). Each node is either used to make a decision (known as decision node) or represent an outcome (known as leaf node).

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g., whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

A decision tree consists of three types of nodes:

- **Decision nodes** – typically represented by squares
- **Chance nodes** – typically represented by circles
- **End nodes** – typically represented by triangles

Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision tree is as a descriptive means for calculating conditional probabilities.

Among decision support tools, decision trees (and influence diagrams) have several advantages. Decision trees:

- Are simple to understand and interpret. People are able to understand decision tree models
- after a brief explanation.
- Help determine worst, best and expected values for different scenarios.

Disadvantages of decision trees:

- They are unstable, meaning that a small change in the data can lead to a large change in
- the structure of the optimal decision tree.
- They are often relatively inaccurate. Many other predictors perform better with similar
- data. This can be remedied by replacing a single decision tree with a random forest of
- decision trees, but a random forest is not as easy to interpret as a single decision tree.

The project has been developed using: C#

ABSTRACT

"Decision Tree Simulation" is a project which uses ID3 Algorithm for building decision tree and find best attribute. The algorithm builds a decision tree from a sample of data. The training phase results in a decision tree, which will be used to classify future input data. Every leaf represents a value of the result class and a non-leaf node represents a decision node. The main idea behind the implementation is to take the dataset and then break it into smaller pieces until the problem is small enough to be solved. This section will describe how this was achieved and will give an overview about the flow in the code. At the start of the program, the user can either enter data by hand or import a .csv file or text file with the training data. Importing the data is pretty straight forward. The first row holds the column title and the last column contains the outcome. Entering the data by hand works accordingly. First the user has to declare the number of columns. This number must be an integer and greater than 1. Next the user is prompted to enter the title of every column. Then the user can enter the values for the columns. After the program has received the data for training, either by hand or by importing a text file, the program creates the decision tree. This process is called training or learning.

Once the learning process is complete the user has different options on how to interact with the program. The user can either print the created tree, export entered data or let the program predict the result. Printing the tree and exporting the test data are pretty straight forward. The print function will be explained in more detail in the next section.

If the user wants to get a prediction, the program prompts the user to enter a value for every previously defined column. After that the program will predict the outcome and print which route it took to iterate through the tree. If no valid route was found, an error message will be printed.

ALGORITM USED

- ID3 algorithm is a simple decision tree learning algorithm developed by Ross Quinlan (1983) This algorithm uses the greedy search technique on a given dataset, to test each attribute at every tree node.
- We used this metric to minimize the depth of the tree (minimize asked questions).
- It is a function that measure which questions provide the most balanced splitting the information gain measures how well a given attribute separates training examples into class labels.
- Information gain is expected reduction of entropy related to specific attribute when we split a decision tree node. The information gain Gain (S, A) of example set S, on attribute A is Gain (S, A) =Entropy(S)-∑ ((|Sv| / |S|) * Entropy (Sv))

ALGORITHM

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller subsets with increase in depth of tree. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor is called root node. Decision trees can handle both categorical and numerical data.

Types of decision trees

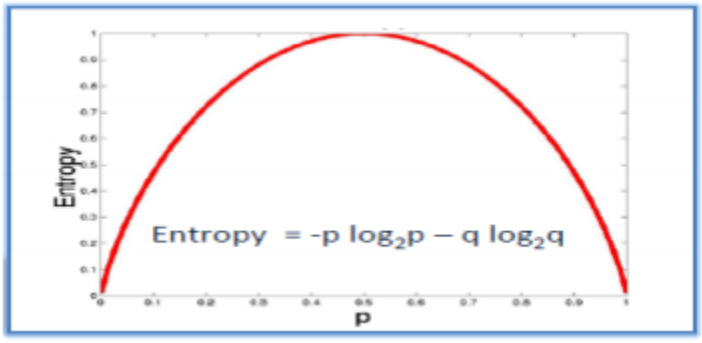
- Categorical Variable Decision Tree: Decision Tree which has categorical target variable then it called as categorical variable decision tree.
- Continuous Variable Decision Tree: Decision Tree which has continuous target variable then it is called as Continuous Variable Decision Tree.

ID3

The core algorithm for building decision trees is called ID3. Developed by J. R. Quinlan, this algorithm employs a top-down, greedy search through the space of possible branches with no backtracking. ID3 uses Entropy and Information Gain to construct a decision tree.

Entropy

A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is equally divided then it has entropy of one.



a) Calculate Entropy (Amount of uncertainty in dataset):

$$Entropy = \frac{-p}{p+n} log_2(\frac{p}{p+n}) - \frac{n}{p+n} log_2(\frac{n}{p+n})$$

b) Calculate Average Information:

$$I(Attribute) = \sum \frac{p_i + n_i}{p+n} Entropy(A)$$

Information Gain: The information gain is based on the decrease in entropy after a data-set is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

Step 1: Calculate entropy of the target.
Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy
Gain (T, X) =Entropy(T) - Entropy (T, X)
Step 3: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.
Step 4a: A branch with entropy of 0 is a leaf node.
Step 4b: A branch with entropy more than 0 needs further splitting.
Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

HARDWARE/SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENT

- Intel(R) Core (TM) i5-4440 CPU @ 3.10GHz
- 8 GB DDR3 RAM.
- 10 GB Hard Disk

SOFTWARE REQUIREMENT:

- **Operating System:** Windows 7 or higher
- **Software Used:** Visual Studio 2019
- **Programming Language:** C#

CODE

Main.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows.Forms;

namespace ID3
{
   public partial class Main : Form
   {
      List<Attribute> Attributes = new List<Attribute>();
      DecisionTreeID3 DTID3;
      List<List<string>> Examples = new List<List<string>>();
      int Height, Width = 0;

      public Main()
      {
         InitializeComponent();
         this.StartPosition = FormStartPosition.CenterScreen;
      }
      private void btnnew_Click(object sender, EventArgs e)
      {
         rtxresult.Clear();
         dgvmain.Columns.Clear();
         btnRun.Enabled = false;
         Attributes.Clear();
         Width = 0;
         Bitmap imageDelete = new Bitmap(pbxpaint.Width, pbxpaint.Height);
         pbxpaint.Image = imageDelete;
      }
      private void btnlearn_Click(object sender, EventArgs e)
      {
         Examples.Clear();
         for (int i = 0; i < dgvmain.Rows.Count - 1; i++)
         {
            List<string> example = new List<string>();
            for (int j = 0; j < dgvmain.Columns.Count; j++)
            {
               example.Add(dgvmain.Rows[i].Cells[j].Value.ToString().ToLower());
```

```
            }
            Examples.Add(example);
        }
        List<Attribute> at = new List<Attribute>();
        for (int i = 0; i < Attributes.Count; i++)
        {
            at.Add(Attributes[i]);
        }
        DTID3 = new DecisionTreeID3(Examples, at);
        DTID3.GetTree();
        Height = DTID3.Depth * 200;
        Width = DTID3.Tree.NumberLabel * 100;
        pbxpaint.Invalidate();
        rtxresult.Text = DTID3.Solution;
    }
    private void cbxc_CheckedChanged(object sender, EventArgs e)
    {
    }
    private void pbxpaint_Paint(object sender, PaintEventArgs e)
    {
        if (Width > 0)
        {
            pbxpaint.Width = Width;
            pbxpaint.Height = Height;
            PaintTree(DTID3.Tree, e, 0, 50);
        }
    }
    private void PaintTree(TreeNode tree, PaintEventArgs e, int X, int Y)
    {

        int XStart = X;
        X = (tree.NumberLabel * 100 + 2 * X) / 2 - 50;
        if (tree.Attributes.Name.ToString() != "")
        {
            e.Graphics.FillRectangle(Brushes.Blue, X, Y, tree.Attributes.Name.Length * 15, 30);
            e.Graphics.DrawString(tree.Attributes.Name.ToString(), new Font("Arial", 20),
Brushes.Red, new PointF(X, Y));
        }
        else
        {
            e.Graphics.FillRectangle(Brushes.Green, X + 25, Y, tree.Attributes.Label.Length * 20,
30);
            e.Graphics.DrawString(tree.Attributes.Label, new Font("Arial", 20), Brushes.Yellow,
new PointF(X + 25, Y));
```

```
        }
        int XEndA;
        for (int i = 0; i < tree.Attributes.Value.Count; i++)
        {
            XEndA = tree.Childs[i].NumberLabel * 100 + XStart;
            int XA = (XStart + XEndA) / 2 - 50;
            e.Graphics.DrawLine(new Pen(Brushes.Black, 2), X + 50, Y + 30, XA + 50, Y + 100);
            e.Graphics.DrawString(tree.Attributes.Value[i].ToString(), new Font("Arial", 20),
Brushes.Blue, new PointF(XA, Y + 100));
            e.Graphics.DrawLine(new Pen(Brushes.Black, 2), XA + 50, Y + 130, XA + 50, Y +
200);
            PaintTree(tree.Childs[i], e, XStart, Y + 200);
            XStart = XEndA;
        }
    }
    private void btnloaddata_Click(object sender, EventArgs e)
    {
        dgvmain.Columns.Clear();
        rtxresult.Clear();
        Attributes.Clear();
        OpenFileDialog OpenDiag = new OpenFileDialog();
        OpenDiag.InitialDirectory = Application.StartupPath;
        OpenDiag.DefaultExt = ".txt";
        OpenDiag.Filter = "Text documents (.txt)|*.txt";
        if (OpenDiag.ShowDialog() == DialogResult.OK)
        {
            StreamReader sr = new StreamReader(OpenDiag.FileName);
            string line = "";
            if ((line = sr.ReadLine()) != null)
            {
                string[] attributeName = line.Trim().ToLower().Split('\t').ToArray();
                for (int i = 0; i < attributeName.Length - 1; i++)
                {
                    Attribute temp = new Attribute();
                    temp.Name = attributeName[i];
                    Attributes.Add(temp);
                    DataGridViewTextBoxColumn column = new DataGridViewTextBoxColumn();
                    column.HeaderText = attributeName[i].ToUpper();
                    column.Name = attributeName[i].ToUpper();
                    dgvmain.Columns.Add(column);
                }
                DataGridViewTextBoxColumn columnend = new DataGridViewTextBoxColumn();
                columnend.HeaderText = attributeName[attributeName.Length - 1].ToUpper();
                columnend.Name = attributeName[attributeName.Length - 1].ToUpper();
```

```
                dgvmain.Columns.Add(columnend);
                btnRun.Enabled = true;
            }
            while ((line = sr.ReadLine()) != null)
            {
                line = line.Trim();
                string[] value = line.Trim().ToLower().Split('\t').ToArray();
                DataGridViewRow dgvr = new DataGridViewRow();
                for (int i = 0; i < value.Length - 1; i++)
                {
                    Attributes[i].AddValue(value[i]);
                }
                string[] value2 = line.Trim().ToUpper().Split('\t').ToArray();
                dgvmain.Rows.Add(value);
            }
            sr.Close();
        }
    }
    private void ID3_Load(object sender, EventArgs e)
    {
        btnRun.Enabled = false;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        About frm = new About();
        frm.Show();
        frm.StartPosition = FormStartPosition.CenterScreen;
    }
  }
}
```

ID3.cs

```
using System;
using System.Collections.Generic;
namespace ID3
{
  class DecisionTreeID3
  {
    List<List<string>> Examples;
    List<Attribute> Attributes;
    public List<string> RuleID3 = new List<string>();
    TreeNode _tree;
    int _depth;
```

```
        string _solution;
        public int ruleCount;
        string Rule;
        internal TreeNode Tree
        {
            get { return _tree; }
            set { _tree = value; }
        }
        public int Depth
        {
            get { return _depth; }
            set { _depth = value; }
        }
        public string Solution
        {
            get { return _solution; }
            set { _solution = value; }
        }
        public DecisionTreeID3(List<List<string>> Examples, List<Attribute> Attributes)
        {
            this.Examples = Examples;
            this.Attributes = Attributes;
            this.Tree = null;
            Depth = 0;
        }
        // entropy calculation
        private double GetEntropy(int Positives, int Negatives)
        {
            if (Positives == 0)
                return 0;
            if (Negatives == 0)
                return 0;
            double Entropy;
            int total = Negatives + Positives;
            double RatePositves = (double)Positives / total;
            double RateNegatives = (double)Negatives / total;
            Entropy = -RatePositves * Math.Log(RatePositves, 2) - RateNegatives *
Math.Log(RateNegatives, 2);
            return Entropy;
        }
        // count Gain(bestat,A);
        private double Gain(List<List<string>> Examples, Attribute A, string bestat)
        {
            double result;
```

```
        int CountPositives = 0;
        int[] CountPositivesA = new int[A.Value.Count];
        int[] CountNegativeA = new int[A.Value.Count];
        int Col = Attributes.IndexOf(A);
        for (int i = 0; i < A.Value.Count; i++)
        {
            CountPositivesA[i] = 0;
            CountNegativeA[i] = 0;
        }
        for (int i = 0; i < Examples.Count; i++)
        {
            int j = A.Value.IndexOf(Examples[i][Col].ToString());
            if (Examples[i][Examples[0].Count - 1] == "1")
            {
                CountPositives++;
                CountPositivesA[j]++;
            }
            else
            {
                CountNegativeA[j]++;
            }
        }
        result = GetEntropy(CountPositives, Examples.Count - CountPositives);
        for (int i = 0; i < A.Value.Count; i++)
        {
            double RateValue = (double)(CountPositivesA[i] + CountNegativeA[i]) /
Examples.Count;
            result = result - RateValue * GetEntropy(CountPositivesA[i], CountNegativeA[i]);
        }
        Solution = Solution + "\n * Gain(" + bestat + "," + A.Name + ") = " + result.ToString();
        return result;
    }
    // ID3 algorithm
    private TreeNode ID3(List<List<string>> Examples, List<Attribute> Attribute, string
bestat)
    {
        Solution = Solution + "---------------------------------    Review " + bestat + "    ----------
----------------";
        if (CheckAllPositive(Examples))
        {
            Solution += "\n All samples assert => Returns the root node with the label 1";
            return new TreeNode(new Attribute("1"));
        }
        if (CheckAllNegative(Examples))
```

```
        {
            Solution += "\n All samples are negative => Returns the root node with the label 0";
            return new TreeNode(new Attribute("0"));
        }
        if (Attribute.Count == 0)
        {
            Solution += "\n Empty properties => Returns the root node with the most common
value ";
            return new TreeNode(new Attribute(GetMostCommonValue(Examples)));
        }
        Attribute BestAttribute = GetBestAttribute(Examples, Attribute, bestat);
        int LocationBA = Attributes.IndexOf(BestAttribute);
        TreeNode Root = new TreeNode(BestAttribute);
        for (int i = 0; i < BestAttribute.Value.Count; i++)
        {
            List<List<string>> Examplesvi = new List<List<string>>();
            for (int j = 0; j < Examples.Count; j++)
            {
                if (Examples[j][LocationBA].ToString() == BestAttribute.Value[i].ToString())
                    Examplesvi.Add(Examples[j]);
            }
            if (Examplesvi.Count == 0)
            {
                Solution += "\n Empty properties => Returns the root node with the most common
value ";
                return new TreeNode(new Attribute(GetMostCommonValue(Examplesvi)));
            }
            else
            {
                Solution += "\n";
                Attribute.Remove(BestAttribute);
                Root.AddNode(ID3(Examplesvi, Attribute, BestAttribute.Value[i]));
            }
        }
        return Root;
    }
    // Get the property with the highest Gain
    private Attribute GetBestAttribute(List<List<string>> Examples, List<Attribute>
Attributes, string bestat)
    {
        double MaxGain = Gain(Examples, Attributes[0], bestat);
        int Max = 0;
        for (int i = 1; i < Attributes.Count; i++)
        {
```

```
            double GainCurrent = Gain(Examples, Attributes[i], bestat);
            if (MaxGain < GainCurrent)
            {
                MaxGain = GainCurrent;
                Max = i;
            }
        }
        Solution = Solution + "\n\t=> We choose the biggest Gain : " + Attributes[Max].Name;
        return Attributes[Max];
    }
    // take the most common value of the target set
    private string GetMostCommonValue(List<List<string>> Examples)
    {
        int CountPositive = 0;
        for (int i = 0; i < Examples.Count; i++)
        {
            if (Examples[i][Examples[0].Count - 1] == "1")
                CountPositive++;
        }
        int CountNegative = Examples.Count - CountPositive;
        string Label;
        if (CountPositive > CountNegative)
            Label = "1";
        else
            Label = "0";
        Solution = Solution + " là " + Label;
        return Label;
    }
    // checks if all sets are positive
    private bool CheckAllPositive(List<List<string>> Examples)
    {
        for (int i = 0; i < Examples.Count; i++)
        {
            if (Examples[i][Examples[0].Count - 1].ToString() == "0")
                return false;
        }
        return true;
    }
    // Check if all episodes are Negative
    private bool CheckAllNegative(List<List<string>> Examples)
    {
        for (int i = 0; i < Examples.Count; i++)
        {
            if (Examples[i][Examples[0].Count - 1] == "1")
```

```
            return false;
        }
        return true;
    }
    // tree construction
    public void GetTree()
    {
        Solution = "";
        List<Attribute> at = new List<Attribute>();
        for (int i = 0; i < Attributes.Count; i++)
        {
            at.Add(Attributes[i]);
        }
        Tree = ID3(Examples, at, "S");
        Depth = GetDepth(Tree);
    }
    // Find the law
    public void SearchRule(TreeNode Rule)
    {
        if (Rule.Attributes.Value.Count == 0)
        { }
        else
        {
            string temp1 = "";
            _solution += Rule.Attributes.Name + " = ";
            temp1 += _solution + " ";
            for (int i = 0; i < Rule.Attributes.Value.Count; i++)
            {
                string temp2 = "";
                temp2 = temp1 + Rule.Attributes.Value[i] + ", ";
                if (Rule.Childs[i].Attributes.Value.Count == 0)
                {
                    ruleCount++;
                    _solution = temp2 + "} THEN { " + Rule.Childs[i].Attributes.Label + "} ";
                    RuleID3.Add(_solution);
                }
                else
                {
                    if (Rule.Attributes.Value.Count == 0)
                    {
                        SearchRule(Rule.Childs[i]);
                    }
                    else
                    {
```

```
                _solution = temp2;
                SearchRule(Rule.Childs[i]);
            }
        }
    }
}
public void GetRule(TreeNode tree)
{
    _solution = "";
    Rule += " WITHDRAWAL FROM DECISION PLANT ID3 \n\n";
    SearchRule(tree);
    for (int i = 0; i < ruleCount; i++)
        Rule += " Rule [" + i + "]: IF {" + RuleID3[i] + "\n";
    Rule += "\n Total Law : " + ruleCount;
    ruleCount = 0;
}
// Decision Tree Optimization.
public bool CheckAllLabelNegative(TreeNode tree)
{
    int test = 0;
    string temp;
    temp = "No";
    for (int i = 0; i < tree.Attributes.Value.Count; i++)
    {
        if (tree.Childs[i].Attributes.Label == temp)
            test++;
    }
    if ((test > 1) && (test == tree.Attributes.Value.Count))
        return true;
    else
        return false;
}
// get the depth of the tree
private int GetDepth(TreeNode tree)
{
    int depth;
    if (tree.Childs.Length == 0)
        return 1;
    else
    {
        depth = GetDepth(tree.Childs[0]);
        for (int i = 1; i < tree.Childs.Length; i++)
        {
```

```
                int depthchild = GetDepth(tree.Childs[i]);
                if (depth < depthchild)
                    depth = depthchild;
            }
            depth++;
        }
        return depth;
    }
  }
}
```

Attributes.cs

```
using System.Collections.Generic;

namespace ID3
{
   class Attribute
   {
      List<string> _value;
      string _name;
      string _label;
      public List<string> Value
      {
         get { return _value; }
         set { _value = value; }
      }
      public string Name
      {
         get { return _name; }
         set { _name = value; }
      }
      public string Label
      {
         get { return _label; }
         set { _label = value; }
      }
      public Attribute()
      {
         this.Name = "";
         this.Label = "";
         this.Value = new List<string>();
      }
      public Attribute(List<string> Value, string Name)
```

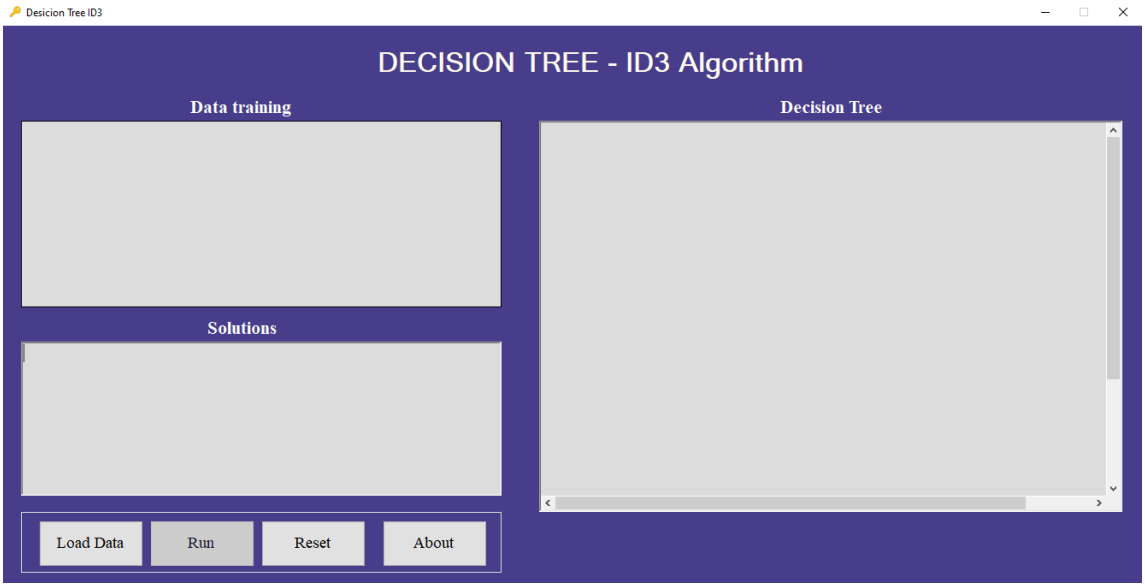
```
        {
            this.Value = Value;
            this.Name = Name;
            this.Label = "";
        }
        public Attribute(string Label)
        {
            this.Label = Label;
            this.Name = string.Empty;
            Value = new List<string>();
        }
        public void AddValue(string Value)
        {
            if (!_value.Contains(Value))
                _value.Add(Value);
        }
    }
}
```

Treenode.cs

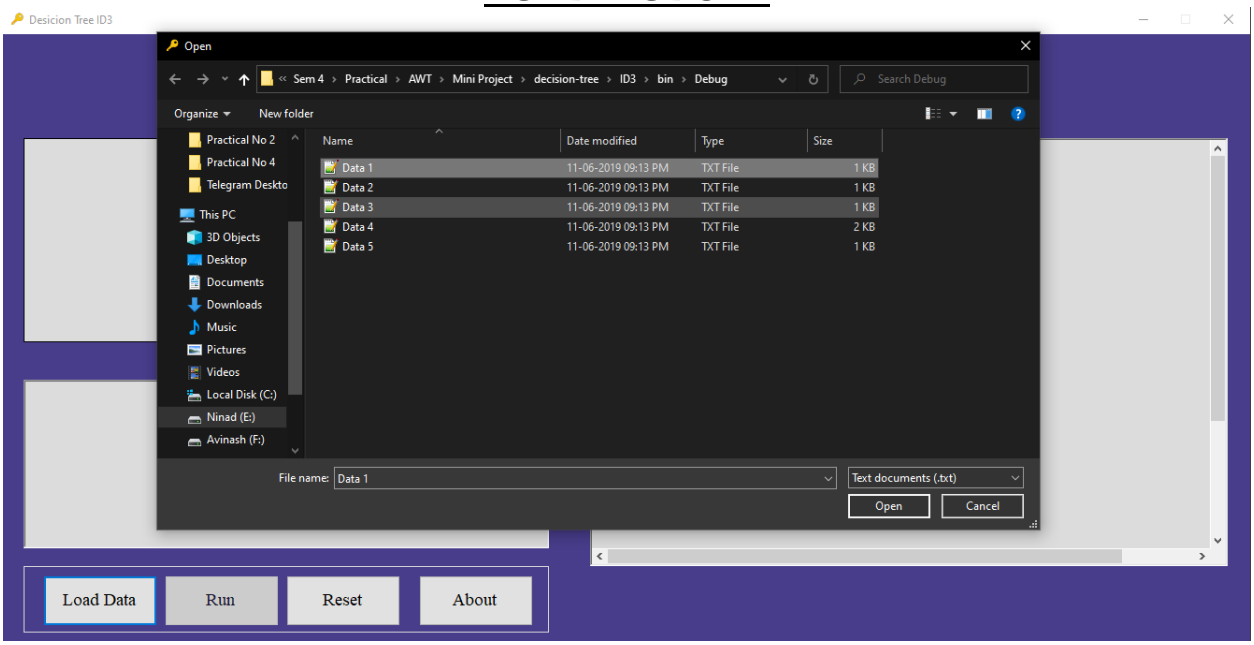
```
namespace ID3
{
    class TreeNode
    {
        Attribute attributes;
        TreeNode[] childs;
        int n;
        int _numberLabel;
        internal Attribute Attributes
        {
            get { return attributes; }
            set { attributes = value; }
        }
        internal TreeNode[] Childs
        {
            get { return childs; }
            set { childs = value; }
        }
        public int NumberLabel
        {
            get { return _numberLabel; }
            set { _numberLabel = value; }
        }
```

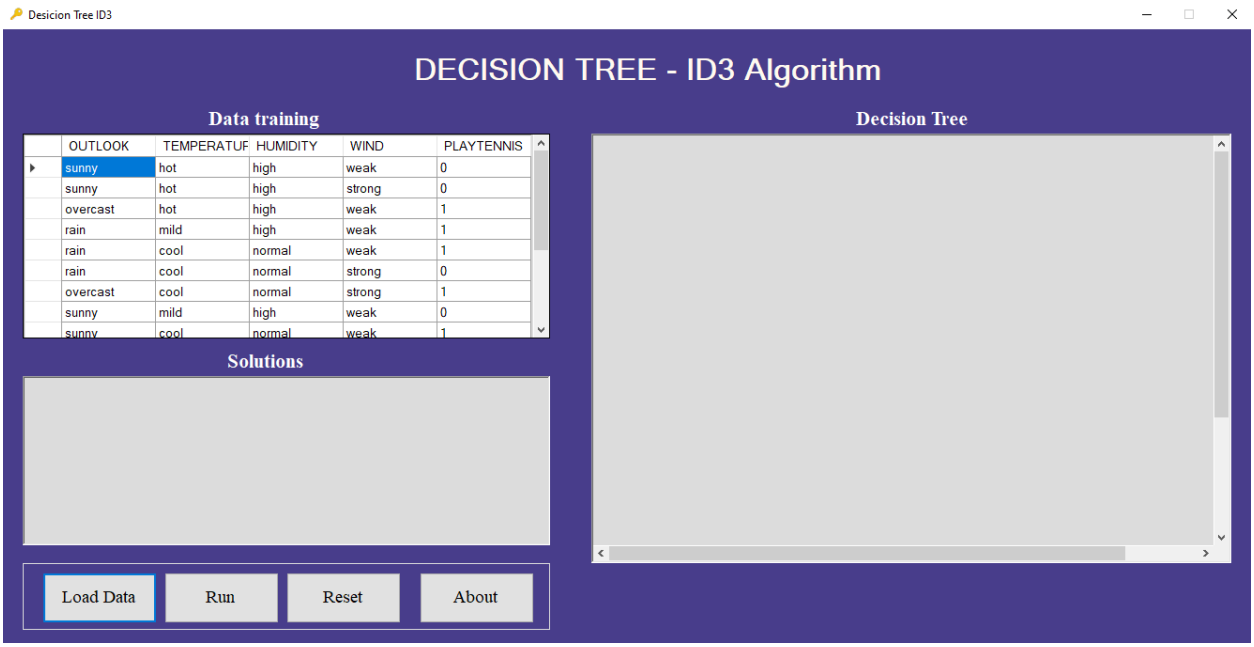
```
        public TreeNode(Attribute Attributes)
        {
            this.Attributes = Attributes;
            this.Childs = new TreeNode[Attributes.Value.Count];
            n = 0;
            for (int i = 0; i < Attributes.Value.Count; i++)
            {
                Childs[i] = null;
            }
            if (Attributes.Value.Count == 0)
                NumberLabel = 1;
            else
                NumberLabel = 0;
        }
        public void AddNode(TreeNode Child)
        {
            if (n < Childs.Length)
            {
                Childs[n] = Child;
                NumberLabel = NumberLabel + Child.NumberLabel;
            }
            n++;
        }
    }
}
```

(SCREENSHOT)
MAIN SCREEN

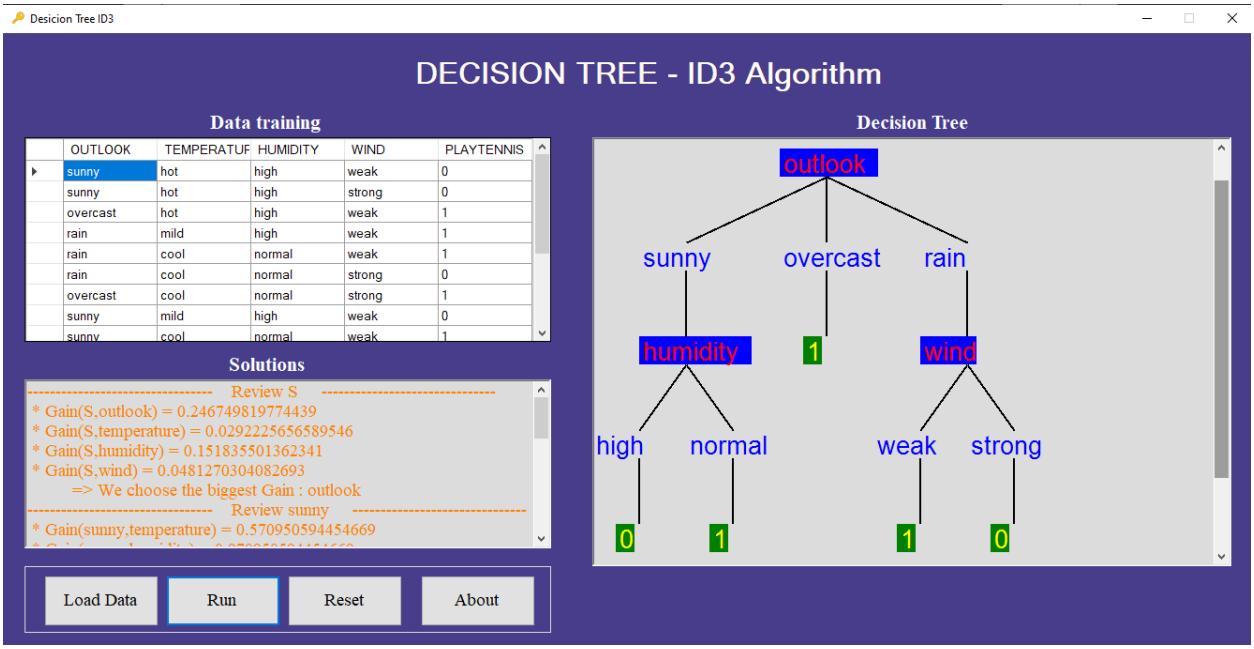


LOAD MODULE

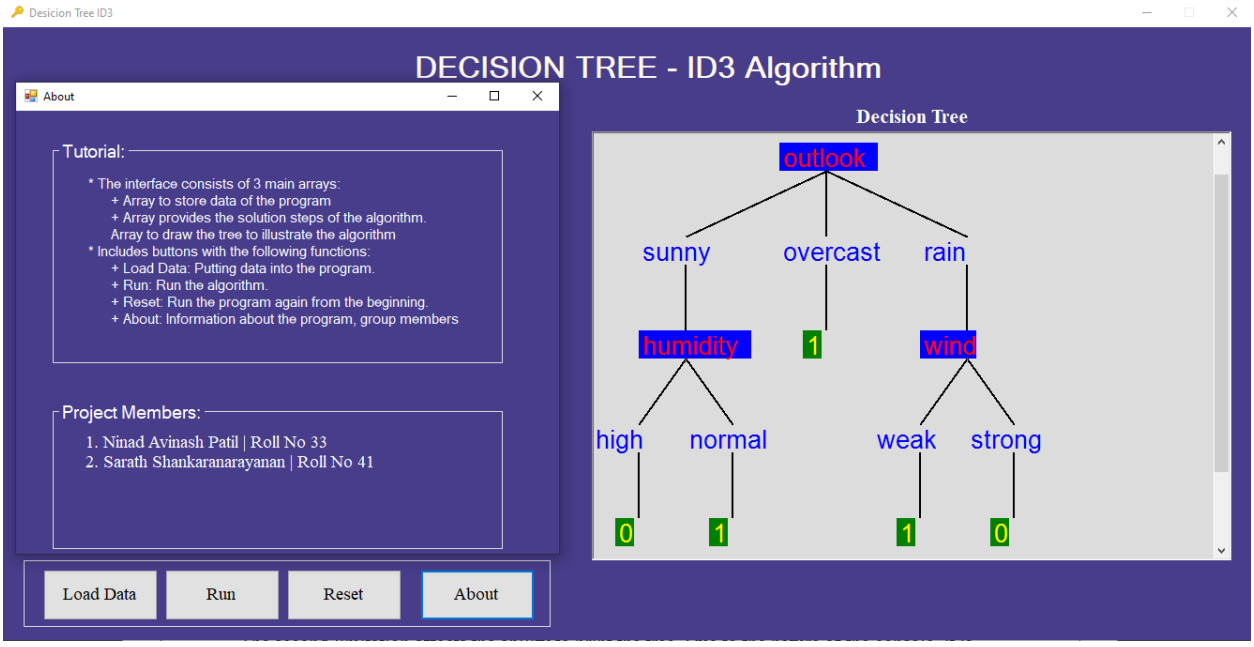

Desicion Tree ID3
Open
Sem 4 › Practical › AWT › Mini Project › decision-tree › ID3 › bin › Debug
Search Debug
Organize
New folder
Practical No 2
Practical No 4
Telegram Deskto
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Ninad (E:)
Avinash (F:)
Name
Date modified
Type
Size
Data 1 11-06-2019 09:13 PM TXT File 1 KB
Data 2 11-06-2019 09:13 PM TXT File 1 KB
Data 3 11-06-2019 09:13 PM TXT File 1 KB
Data 4 11-06-2019 09:13 PM TXT File 2 KB
Data 5 11-06-2019 09:13 PM TXT File 1 KB
File name: Data 1
Text documents (.txt)
Open
Cancel
Load Data
Run
Reset
About


Desicion Tree ID3
DECISION TREE - ID3 Algorithm
Data training
Decision Tree
OUTLOOK TEMPERATUF HUMIDITY WIND PLAYTENNIS
sunny hot high weak 0
sunny hot high strong 0
overcast hot high weak 1
rain mild high weak 1
rain cool normal weak 1
rain cool normal strong 0
overcast cool normal strong 1
sunny mild high weak 0
sunny cool normal weak 1
Solutions
Load Data
Run
Reset
About

RUN MODULE



ABOUT MODULE



LIMITATIONS AND FUTURE ENHANCEMENT

- The first limitation is that the last column of the training data, the result column, can only contain two different values. These values should be 1 and 0. If the result values are two different values, but not 1 and 0, the program will work but the leaves are not identified as such and therefore printed in a wrong colour.
- The second limitation affects the ability to print the tree. Due to the nature of the console, it is not possible to create a print function which can print all sizes and variations of a tree in a user-friendly way. Therefore, the approach with different colours, separators and casings were chosen.

FUTURE SCOPE OF THE PROJECT

- To add other algorithms
- Future Scope is to add different values to the end column other than 1 and 0

CONCLUSION

- So, we have created a project based on algorithm ID3.
- We have tested this project with different dataset and the result we are getting is in good accuracy.
- So, this project using ID3 algorithm can predict the possibility using the dataset.
- According to the data it will generate a decision tree and we can predict the possibility according to the decision tree.

REFERENCE

Web Reference:

- https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
- https://www.javatpoint.com/decision-tree-induction
- https://www.guru99.com/c-sharp-windows-forms-application.html

Textbook Reference:

- Programming in C# by E Balagurusamy.
- Data Mining and Business Intelligence by Abhishek Swaroop, Devesh Agarwal.
- Data Mining Concepts and Techniques 3rd Edition by Jinesh Melvin.