



UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

DEPARTMENT OF ELECTRICAL, ELECTRONIC
AND COMPUTER ENGINEERING

EAI 320 - INTELLIGENT SYSTEMS

EAI 320 - Practical Assignment 5 Report

Author:

MOHAMED AMEEN OMAR

Student number:

u16055323

March 30, 2018

Contents

1	Introduction	1
2	Problem Definition	2
3	Implementation and Methodology	4
3.1	Node Class	4
3.2	The ID3 class	4
4	Results	6
5	Discussion	13
6	Conclusion	14
7	Appendix A: Python Code	15
8	Bibliography	24

List of Figures

1	Sample data given in the practical specification	2
2	The test cases given to students in the practical specification	3
3	Example of the format for a single test case in a *.csv file	3
4	Console output showing the building of the tree with the data set given in the practical specification	6
5	Partial console output showing the building of the tree with any data set in a *.csv file (that is not the program directory)	7
6	Partial console output when multiple inputs are tested	7
7	Partial console output illustrating the input validation for the program .	8
8	Console output showing the path followed in the tree to find the decision for testCase1.csv	8
9	Console output showing the path followed in the tree to find the decision for testCase2.csv	9
10	Console output showing the path followed in the tree to find the decision for testCase3.csv	9
11	Console output showing the path followed in the tree to find the decision for myTest.csv	10
12	Console output showing the path followed in the tree to find the decision for myTest2.csv	11
13	Initial Entropy values for each attribute for the given data set	12
14	Diagrammatic representation of the decision tree for the data set provided in the practical specification	12
15	Program Source Code 1 of 9 [Python]	15
16	Program Source Code 2 of 9 [Python]	16
17	Program Source Code 3 of 9 [Python]	17
18	Program Source Code 4 of 9 [Python]	18
19	Program Source Code 5 of 9 [Python]	19
20	Program Source Code 6 of 9 [Python]	20
21	Program Source Code 7 of 9 [Python]	21

22	Program Source Code 8 of 9 [Python]	22
23	Program Source Code 9 of 9 [Python]	23

1 Introduction

A decision tree is a data structure that is used to model or predict the outcome or action of a certain situation. A decision tree is a method for approximating or predicting the output from a set of discrete-valued inputs or known observations. The decision is constructed from a set of known data (the sample set) and is then used to classify the outcome for a new set of data - unknown data.

A decision tree can be represented as a series of if and then statements, where each node in the tree is a category whose attribute value is tested and based off of this test, dictates which path along the tree the input should follow. An example of a decision tree can be found in figure 14.

The applications of decision tree learning (the use of decision trees in modelling the outcome based off of a finite number of observations) encompasses a large variety of fields including data mining, statistics and machine learning. The attraction to the use of decision trees stems from its ability to split up large data sets into smaller subsets. This is the reason why decision trees work so well with large data sets that have many different attributes (Big Data).

The ID3, or Iterative Dichotomiser 3, algorithm is a recursive decision tree building algorithm that classifies the different attributes of a given data set using the Information gain or the Entropy of a given attribute. The Information Gain of an attribute is the amount of useful information that can be extracted about the data set when evaluating the given attribute. The Entropy of an attribute is lack of predictability that an attribute provides with regards to the data set.

The ID3 recursive algorithm builds the decision tree by first splitting up the attribute with the highest Information gain and thus the lowest entropy.

2 Problem Definition

Students were tasked with implementing and testing the Iterative Dichotomiser 3 (ID3) decision tree building algorithm for the sample data given in figure 1. It was up to students to research further regarding the details of Decision Trees as well as the ID3 algorithm.

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
X_1	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>0-10</i>	<i>T</i>
X_2	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>30-60</i>	<i>F</i>
X_3	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>Some</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>T</i>
X_4	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>10-30</i>	<i>T</i>
X_5	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>French</i>	<i>>60</i>	<i>F</i>
X_6	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Italian</i>	<i>0-10</i>	<i>T</i>
X_7	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>0-10</i>	<i>F</i>
X_8	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>Some</i>	<i>\$\$</i>	<i>T</i>	<i>T</i>	<i>Thai</i>	<i>0-10</i>	<i>T</i>
X_9	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>Full</i>	<i>\$</i>	<i>T</i>	<i>F</i>	<i>Burger</i>	<i>>60</i>	<i>F</i>
X_{10}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$\$\$</i>	<i>F</i>	<i>T</i>	<i>Italian</i>	<i>10-30</i>	<i>F</i>
X_{11}	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>None</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Thai</i>	<i>0-10</i>	<i>F</i>
X_{12}	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>Full</i>	<i>\$</i>	<i>F</i>	<i>F</i>	<i>Burger</i>	<i>30-60</i>	<i>T</i>

Figure 1: Sample data given in the practical specification

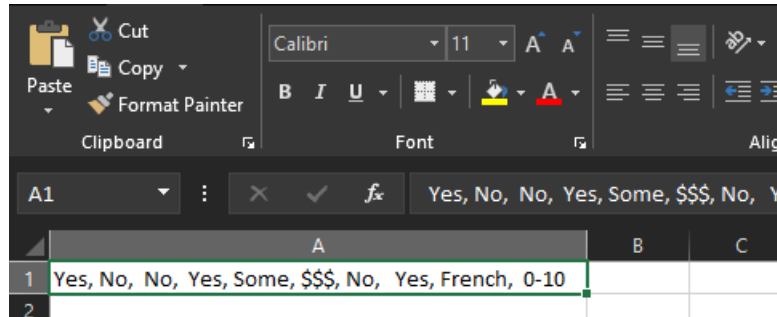
The ID3 algorithm students were tasked to implement needed to be able to handle any data set presented in a particular format. Thus students needed to make their algorithm as generic and abstract as possible.

The data was given to students in a *.csv file which a comma separated values file. Thus each attribute value was separated by a comma and each row in the *.csv represented one example.

$$\begin{aligned}
X_1 &= \{Alt = No, Bar = No, Fri = Yes, \\
&\quad Hun = Yes, Pat = Full, Price = \$\$, \\
&\quad Rain = Yes, Res = No, Type = Thai, Est = 10 - 30\} \\
X_2 &= \{Alt = Yes, Bar = No, Fri = No, \\
&\quad Hun = Yes, Pat = Full, Price = \$\$, \\
&\quad Rain = Yes, Res = No, Type = Italian, Est = 30 - 60\} \\
X_3 &= \{Alt = Yes, Bar = No, Fri = Yes, \\
&\quad Hun = Yes, Pat = Some, Price = \$, \\
&\quad Rain = No, Res = No, Type = Burger, Est = 0 - 10\}
\end{aligned}$$

Figure 2: The test cases given to students in the practical specification

The test cases provided to students in order to test their implementation is given in figure 2. This information was then put into a *.csv file in order to allow it to be imported into the program for analysis. An example of a single test case formatted into a *.csv is shown in figure 3.



The screenshot shows a spreadsheet application with a dark theme. The 'Clipboard' tab is active, showing options like Cut, Copy, Paste, and Format Painter. The 'Font' tab is also visible, showing font settings like Calibri, size 11, and bold/italic/underline options. The spreadsheet grid shows a single row with the following data: 'Yes, No, No, Yes, Some, \$\$\$, No, Yes, French, 0-10'. The row is highlighted in green, and the column headers are A, B, and C.

	A	B	C
1	Yes, No, No, Yes, Some, \$\$\$, No, Yes, French, 0-10		
2			

Figure 3: Example of the format for a single test case in a *.csv file

3 Implementation and Methodology

In order to implement the ID3 decision tree building algorithm and allow for the tree to be tested, three classes were implemented. Namely, the Stack class - used to implement a stack data structure in python, the Node class and the ID3 class. The source code can be found in Appendix A.

3.1 Node Class

In order to implement the ID3 decision tree building algorithm a node class was implemented to facilitate the information for each attribute in the tree.

The node class contains multiple member variables including:

1. *category* - The name of the category to which this node belongs if it is a category node or an attribute node.
2. *value* - The attribute that this node represents for the category to which it belongs. Or it is the decision or solution if the node is a decision node.
3. *parent* - a reference to the parent of the current node.
4. *children* - a Python list of references to all the child nodes of the current node.
5. *subset* - is a list of python dictionaries that represent the data that the *subtree* with this node as the root is made up of.
6. *isCategoryNode* - a boolean which indicates whether the node is a category node, meaning whether it's children represent attribute values for the category to which they belong. The name of the category is store in the *category* member variable.
7. *isDecisionNode* - a boolean which indicates whether the node is a decision node, meaning whether it's value is a solution to a given sample (it is a leaf in the decision tree).

3.2 The ID3 class

The ID3 class is where the actual implementation of the ID3 algorithm and the running of the program takes place.

The user is first prompted as whether they would like to use the default data set provided in the practical specification to construct the decision tree or whether they would like to input their own sample data, this is illustrated in figures 4 and 5. The data needs to be given in a *.csv and it is then converted into a list of dictionaries using the functions provided in the *csv* Python library Once the data has been captured by the program, the decision tree is built using the *build()* member function.

The *build()* member function is a recursive algorithm that constructs the tree according

to the rules dictated by the ID3 algorithm. It first checks the root of the tree to see if it has been created, if not it instantiates a new node and assigns it as the root.

Once the tree has been constructed the user can then input samples for which the result is outputted to the console. Multiple samples can be tested with a single run of the program and once there are no samples to be tested, the program outputs a summary of all the results found by the tree during the current execution of the program. This can be seen in figures 6 to 12.

The initial entropy values for each attribute is also shown in 13 in order to prove that the entropy calculation implemented in the *getEntropy()* member function does work. The entropy values are marginalised over the conditional probability of an attribute, to find the overall entropy value.

The ID3 algorithm

The ID3 algorithm is a recursive implementation wherein the *getEntropy()* and the *getLowestEntropy()* member functions are used in order to determine the best category to split the data. Once this has been determined, the attribute values for the category at which the data will be split is determined and a child node for each attribute value is created. The data is then split into smaller subsets for each category attribute value child created. Then for each child that is created, the algorithm determines if the subset is pure, meaning that the subset only contains a single solution for all the data it contains. If true, the algorithm creates a decision node for that subtree. If it is not pure, the algorithm is run on the child and the data is split again. This process continues until the entire tree has been constructed.

4 Results

The figures below show the console output of the ID3 algorithm and the path followed in order to search for a solution for the given samples.

```
Would you like to build a decision tree for the data in restaurant(1).csv? (Please input Y or N)
y
Building Tree...
Tree has been built

The Tree has:
27 Nodes.
6 Category Nodes.
13 Attribute Value Nodes.
8 Decision Nodes.

Would you like to input a test case?
y
Please enter the name of the file for which you would like to know the outcome.
Please include the file extension (.csv) as well

Test file Name: testCase1.csv
Testing attributes in file: testCase1.csv with the sample data
Retrieving Decision....

At a Category Node.
The category being compared is: Pat

At an attribute value Node.
Attribute is : Full

At a Category Node.
The category being compared is: Fri

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Alt

At an attribute value Node.
Attribute is : No

At a decision Node
Congratulations we have found a decision
Based off of the sample data, the final decision for this test case is predicted to be: No

Would you like to test another file?
n
A reminder, the decisions for the input test cases were:
Decision 1 was: No
The program has ended
```

Figure 4: Console output showing the building of the tree with the data set given in the practical specification

```
Would you like to build a decision tree for the data in restaurant(1).csv? (Please input Y or N)
n
Building Tree...
Please note, the data needs to be in a csv file with the target value or final decision being the 1

Please input the name of the file name to be used to build the tree.
test.csv

How many attributes does the data consist of?
2

Please enter attribute number 1
hello

Please enter attribute number 2
bye
hello
bye
Traceback (most recent call last):
```

Figure 5: Partial console output showing the building of the tree with any data set in a *.csv file (that is not the program directory)

```
Would you like to test another file?
n
A reminder, the decisions for the input test cases were:
Decision 1 was: No
Decision 2 was: No
Decision 3 was: Yes
The program has ended
```

Figure 6: Partial console output when multiple inputs are tested

```

Would you like to build a decision tree for the data in restaurant(1).csv? (Please input Y or N)
p

You entered p which is invalid, please input "Y" or "N"
y
Building Tree...
Tree has been built

The Tree has:
27 Nodes.
6 Category Nodes.
13 Attribute Value Nodes.
8 Decision Nodes.

Would you like to input a test case?
l

You entered l which is invalid, please input "Y" or "N"
y
Please enter the name of the file for which you would like to know the outcome.
Please include the file extension (.csv) as well

Test file Name: |

```

Figure 7: Partial console output illustrating the input validation for the program

```

Test file Name: testCase1.csv
Testing attributes in file: testCase1.csv with the sample data
Retrieving Decision....

At a Category Node.
The category being compared is: Pat

At an attribute value Node.
Attribute is : Full

At a Category Node.
The category being compared is: Fri

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Alt

At an attribute value Node.
Attribute is : No

At a decision Node
Congratulations we have found a decision
Based off of the sample data, the final decision for this test case is predicted to be: No

```

Figure 8: Console output showing the path followed in the tree to find the decision for testCase1.csv

```
Test file Name: testCase2.csv
Testing attributes in file:  testCase2.csv with the sample data
Retrieving Decision....

At a Category Node.
The category being compared is:  Pat

At an attribute value Node.
Attribute is : Full

At a Category Node.
The category being compared is:  Fri

At an attribute value Node.
Attribute is : No

At a decision Node
Congratulations we have found a decision
Based off of the sample data, the final decision for this test case is predicted to be:  No
```

Figure 9: Console output showing the path followed in the tree to find the decision for testCase2.csv

```
Please enter the name of the file for which you would like to know the outcome.
Please include the file extension (.csv) as well

Test file Name: testCase3.csv
Testing attributes in file:  testCase3.csv with the sample data
Retrieving Decision....

At a Category Node.
The category being compared is:  Pat

At an attribute value Node.
Attribute is : Some

At a decision Node
Congratulations we have found a decision
Based off of the sample data, the final decision for this test case is predicted to be:  Yes
```

Figure 10: Console output showing the path followed in the tree to find the decision for testCase3.csv

```
Please enter the name of the file for which you would like to know the outcome.  
Please include the file extension (.csv) as well  
  
Test file Name: myTest.csv  
Testing attributes in file: myTest.csv with the sample data  
Retrieving Decision....  
  
At a Category Node.  
The category being compared is: Pat  
  
At an attribute value Node.  
Attribute is : Some  
  
At a decision Node  
Congratulations we have found a decision  
Based off of the sample data, the final decision for this test case is predicted to be: Yes
```

Figure 11: Console output showing the path followed in the tree to find the decision for myTest.csv

```
Test file Name: myTest2.csv
Testing attributes in file: myTest2.csv with the sample data
Retrieving Decision....

At a Category Node.
The category being compared is: Pat

At an attribute value Node.
Attribute is : Full

At a Category Node.
The category being compared is: Fri

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Alt

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Hun

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Bar

At an attribute value Node.
Attribute is : Yes

At a Category Node.
The category being compared is: Res

At an attribute value Node.
Attribute is : Yes

At a decision Node
Congratulations we have found a decision
Based off of the sample data, the final decision for this test case is predicted to be: No
```

Figure 12: Console output showing the path followed in the tree to find the decision for myTest2.csv

```

The initial entropy values for the given data set:
The Entropy for Alt is 1.0
The Entropy for Bar is 1.0
The Entropy for Fri is 0.9792791603760922
The Entropy for Hun is 0.8042903712002691
The Entropy for Pat is 0.4591479170272448
The Entropy for Price is 0.8042903712002692
The Entropy for Rain is 1.0
The Entropy for Res is 0.9792791603760922
The Entropy for Type is 0.9999999999999999
The Entropy for Est is 0.792481250360578

```

Figure 13: Initial Entropy values for each attribute for the given data set

Figure 14 shows the resulting decision constructed with the sampled data provided in the practical specification.

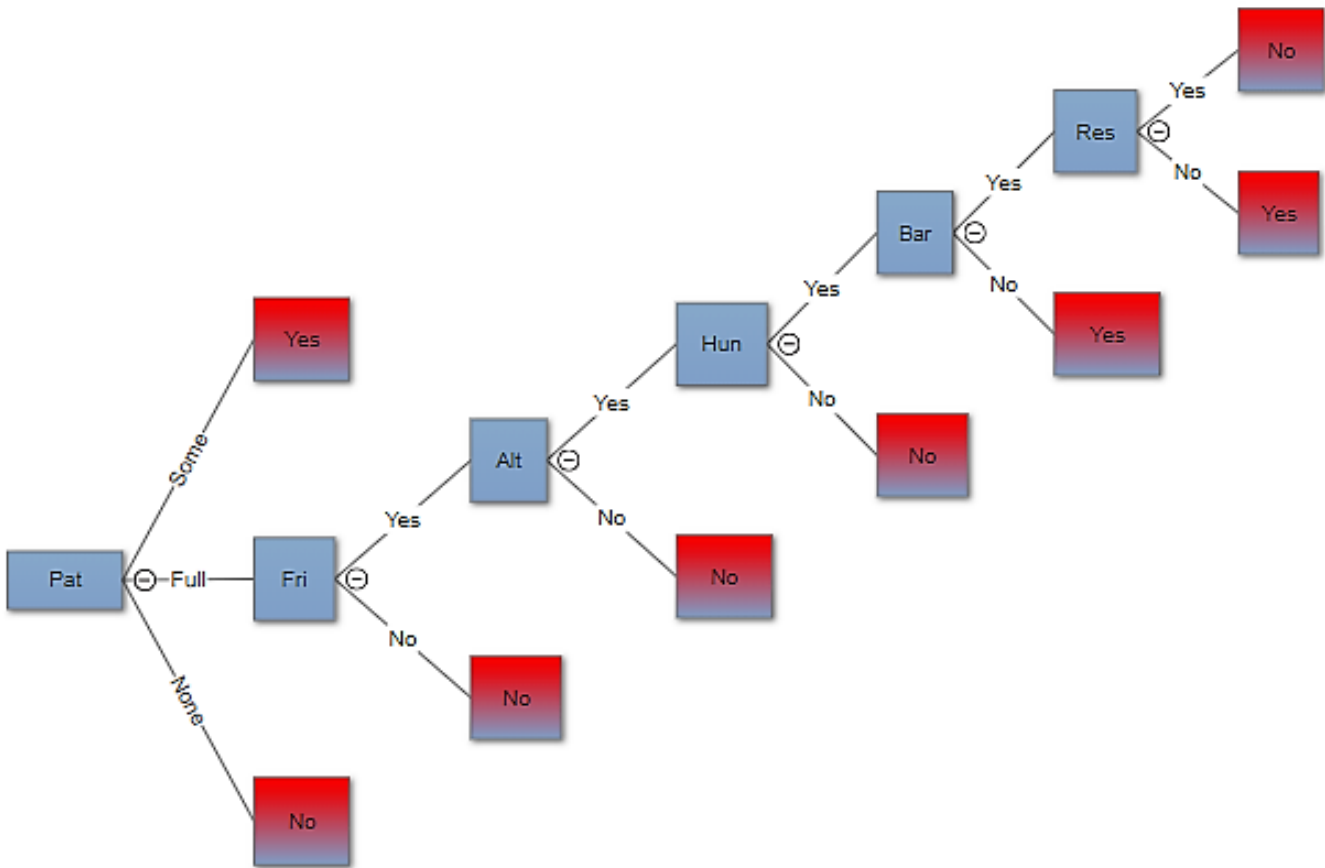


Figure 14: Diagrammatic representation of the decision tree for the data set provided in the practical specification

5 Discussion

The ID3 algorithm is extremely efficient in constructing a decision tree for a given data set. The use of recursion makes it much simpler to implement a tree building function. Once the tree is constructed, the searching for a solution for a given sample is always extremely quick, making finding solutions for large amounts of samples much easier and less tedious as compared to other search strategies.

The tree also does not store all the information it is given, since not all of it is needed in order to find a solution, this is illustrated in 14 and 7. The decision tree only stores 6 Category nodes when the given data set contained a total of 10 Categories. This difference may seem minute, however, with larger data sets that contain any more categories, the storage space saved would be dramatic.

As we can see the entropy value is an extremely powerful metric in order to determine at which category does the data get split in order to reduce the size of the tree and break up the large data set into smaller subsets.

From figure 8 we can see that first the *Pat* category is compared, then the *Fri* category, then *Alt* until finally a decision node is reached which provides a solution of *No* to the sample (X1). Similarly for X2 and X3, the decision tree provides a solution of *No* and *Yes* respectively. The paths followed to find a solution for these samples can be found in figures 9 and 10.

These results given by the decision tree are accurate as it provides a successful inference as to the decision or solution that would be reached given the information provided.

6 Conclusion

The Iterative Dichotomiser 3 (ID3) decision tree building algorithm is an extremely fast algorithm that can easily build the decision tree given a sample data set (the training data) by making use of the attribute information gain. The algorithm makes use of a greedy-divide and conquer strategy wherein the algorithm always chooses the attribute with the lowest Entropy and highest Information gain in order to split the data. The split data enables for faster processing of the data since smaller subsets of data can be processed much faster than one large set of data.

The ID3 algorithm however, does not guarantee the solutions outputted by the decision tree will always be correct. The "correctness" of the solution is very heavily reliant on the type of data that is used to train or build the tree. A decision tree constructed with a large set of sample data and a variety of possible outcomes will output more reliable solutions than a tree constricted with sample data that lacks variety and is small in size. If the training data is large but lacks variety one would observe a scenario termed as *overfitting*, wherein the decision tree will only output accurate solutions for samples that closely resemble or closely relate to the training data used to construct the tree.

The *overfitting* scenario very closely resembles one getting stuck, in a local maxima or minima, a scenario found in search algorithms such as the *Genetic Algorithm* or the *Hill climbing search*. Thus it is of utmost importance the training data used has a large variety of possible outcomes and is of a reasonable size.

The ID3 algorithm successfully implemented, successfully built the tree and was able to successfully predict the outcomes of the given samples.

7 Appendix A: Python Code

```
1 #Mohamed Ameen Omar
  #u16055323
  #EAI PRACTICAL ASSIGNMENT 5
  #2018
import csv
6 import math
import copy

class Stack:
    def __init__(self):
11         self.list = []
    def isEmpty(self):
        if(self.list == []):
            return True
        return False
16 def push(self, item):
    self.list.append(item)
    def pop(self):
        return self.list.pop()
    def peek(self):
21         return self.list[len(self.items)-1]
    def size(self):
        return len(self.list)

class Node:
26 def __init__(self, category = None, parent = None, child = []):
    self.category = category #category
    self.value = None #only assigned if it is a split node
    self.parent = parent #parent of the node
    self.children = child
31 self.subset = None # for the subset for which this node represnts
    self.isCategoryNode = False #if it is a category or a option
    self.isDecisionNode = False #if it is a decision
    def isLeaf(self):
        if(self.child is None):
36             return True
        return False

class ID3:
41 def __init__(self):
    self.root = None
    self.fileName = ""
    self.mainData = None #each index is a dictionary for the data given
    self.numAttributes = 0
    self.categories = ""
46 self.default = None #boolean for the default files
    self.defaultFilename = "restaurant(1).csv"
    self.defaultCategories = ["Alt", "Bar", "Fri", "Hum", "Pat", "
Price", "Rain", "Res", "Type", "Est", "WillWait"]
    self.testedDecisions = []
    self.numNodes = 0
```

Figure 15: Program Source Code 1 of 9 [Python]

```

self.numAttNodes = 0
self.numDecNodes = 0
self.numCatNodes = 0
self.runID3()

#dummy fucntion to do admin work for the program and make it run
def runID3(self):
    anotherFile = input("Would you like to build a decision tree for
the data in restaurant(1).csv? (Please input Y or N)\n")
    while(anotherFile.upper() != "Y" and anotherFile.upper() != "N"):
        string = "You entered"
        string = string + " "
        string= string + anotherFile
        string = string + (" which is invalid , please input \"Y\" or
\"N\"\n")
        anotherFile = input(string)
    print("Building Tree...")
    if(anotherFile.upper() == "Y"):
        self.default = True
        self.buildDefaultTree()
    else:
        self.default = False
        self.buildOtherTree()
    self.countNodes()
    print("Tree has been built")
    print()
    print("The Tree has:")
    print(self.numNodes, end = " ")
    print("Nodes.")
    print(self.numCatNodes, end = " ")
    print("Category Nodes.")
    print(self.numAttNodes, end = " ")
    print("Attribute Value Nodes.")
    print(self.numDecNodes, end = " ")
    print("Decision Nodes.")
    print()
    test = (input("Would you like to input a test case?\n")).upper()
    while(test.upper() != "Y" and test.upper() != "N"):
        string = "You entered"
        string = string + " "
        string= string + test
        string = string + (" which is invalid , please input \"Y\" or
\"N\"\n")
        test = input(string)
    if(test.upper() == "N"):
        print("The program has ended.")
        return
    print("Please enter the name of the file for which you would like
to know the outcome.")
    print("Please include the file extension (.csv) as well")
    testFile = input("Test file Name: ")
    testList = self.readTestFile(testFile)
    print("Testing attributes in file: ", testFile , end = "")
    print(" with the sample data")

```

Figure 16: Program Source Code 2 of 9 [Python]

```

5         while(again.upper() == "Y"):
6             print("Please enter the name of the file for which you would
7             like to know the outcome.")
8             print("Please include the file extension (.csv) as well")
9             testFile = input("Test file Name: ")
10            testList = self.readTestFile(testFile)
11            print("Testing attributes in file: ", testFile, end = "")
12            print(" with the sample data")
13            print("Retrieving Decision....")
14            self.testFunc(testList)
15            again = input("Would you like to test another file?\n")
16            while(again.upper() != "Y" and again.upper() != "N"):
17                string = "You entered"
18                string = string + " "
19                string= string + again
20                string = string + (" which is invalid , please input \"Y\"
21                or \"N\"\\n")
22                again = input(string)
23            if(again.upper() == "N"):
24                print("A reminder, the decisions for the input test cases were
25                : ")
26                count = 1
27                for x in self.testedDecisions:
28                    print("Decision ", count, end = " ")
29                    print("was:", x)
30                    count = count + 1
31                print("The program has ended")
32                return
33
34            #takes in a list of dicnonaries and returns the dicisions
35            def testFunc(self, testList):
36                tempDict = testList[0]
37                myNode = self.root
38                decision = False
39                while(decision == False):
40                    if(myNode.isDecisionNode == True):
41                        decision = True
42                        print()
43                        print("At a decision Node")
44                        print("Congratulations we have found a decision")
45                        print("Based off of the sample data, the final decision
46                        for this test case is predicted to be: ", myNode.value)
47                        self.testedDecisions.append(myNode.value)
48                    elif(myNode.isCategoryNode == True):
49                        print()

```

Figure 17: Program Source Code 3 of 9 [Python]

```

4 print("Retrieving Decision...")
    self.testFunc(testList)
    again = input("Would you like to test another file?\n")
    while(again.upper() != "Y" and again.upper() != "N"):
        string = "You entered"
        string = string + " "
        string= string + again
        string = string + (" which is invalid , please input \"Y\" or
9         \"N\"\\n")
        again = input(string)
print("At a Category Node.")
    print("The category being compared is: ", myNode.category)
    for child in myNode.children:
        if(tempDict[myNode.category] == child.value):
14         myNode = child
    else:
        print()
        print("At an attribute value Node.")
        print("Attribute is :", myNode.value)
19         myNode = myNode.children[0]
#builds the tree for the default given csv in the prac spec
def buildDefaultTree(self):
    self.fileName = self.defaultFilename
    self.categories = self.defaultCategories
24     self.numAttributes = len(self.categories)
    self.readFile(self.fileName, self.categories)
    self.build()

#builds a tree that isnt for the default csv
29 def buildOtherTree(self):
    self.categories = []
    print("Please note, the data needs to be in a csv file with the
target value or final decision being the last value in every row.")
    name = input("Please input the name of the file name to be used to
build the tree.\n")
    self.fileName = name
34     numAttributes = input("How many attributes does the data consist
of?\n")
    y = 1
    while(y <= int(numAttributes)):
        string = "Please enter attribute number " + str(y) + "\n"
        att = input(string)
39         y = y+1
        self.categories.append(att)
    for x in self.categories:
        print(x)
    self.readFile(self.fileName, self.categories)
44     self.build()

```

Figure 18: Program Source Code 4 of 9 [Python]

```

1  #reads a test file
   def readTestFile(self , fileName):
       temp = []
       csvfile = open(fileName)
       cat = copy.deepcopy(self.categories)
6  del cat[-1]
       temp = list(csv.DictReader(csvfile , cat)) #make the DictReader
       iterator a list
       for decision in temp:
           for k,v in decision.items():
               decision[k] = v.replace(" ", "")
11      return temp
#helper to read the file and sort out main data
   def readFile(self , fileName , categories):
       csvfile = open(fileName)
       self.mainData = list(csv.DictReader(csvfile , categories)) #make
the DictReader iterator a list
16      for decision in self.mainData:
           for k,v in decision.items():
               decision[k] = v.replace(" ", "")
       # returns the entropy value for a category within a given data subset
       #name is the name of the catgory who's entropy we are getting
       #"data" is the dicntonary list which we are using to get the entropy
21      def getEntropy(self , name, data):
           if(data is None):
               return 1
           variableNames = self.getVariableValues(data , name) #for the category
we are testing
26      decisions = [] #different decisions that can result
           for x in data:
               new = True
               if(decisions == []):
                   decisions.append(x[self.categories[self.numAttributes-1]])
31      else:
           for y in decisions:
               if(y == x[self.categories[self.numAttributes-1]]):
                   new = False
               if(new == True):
36      decisions.append(x[self.categories[self.numAttributes
-1]])
       # now we have the decision names and the different variable names
       totalOcc = len(data)
       entropy = 0.0
       tempEnt = 0.0
41      for attribute in variableNames:
           tempEnt = 0.0
           attOcc = self.getAttributeOccurrences(attribute , name, data)
           for decision in decisions:
               temp = 0.0
               tempDecOcc = self.getAttDecOccurrences(attribute , name,
46      decision , data)
               if(tempDecOcc == 0):
                   temp = 0

```

Figure 19: Program Source Code 5 of 9 [Python]

```

else:
    temp = (tempDecOcc/attOcc) * math.log2(tempDecOcc/
attOcc)
    tempEnt = tempEnt + temp
    tempEnt = ((-1 * attOcc)/totalOcc)*tempEnt
    entropy = entropy + tempEnt
return entropy
7 #returns the amount of times a attribute value= attribute , is found within
  a category = category
  #within the data dictionary list
  def getAttributeOccurrences(self , attribute ,category ,data):
    occ = 0
    for x in data:
12      if(x[category] == attribute):
        occ = occ +1
    return occ
#takes in a data subset , which is a list of dictionaries , sees if it is
pure
#by checking that the decision for all is the same
17 def isPure(self ,data):
    if(len(data) == 1):
        return True
    decisionIndex = self.categories[self.numAttributes-1]
    decisionKey = data[0][decisionIndex]
22    for decision in data:
        if(decision[decisionIndex] != decisionKey):
            return False
    return True
#returns the name of the category which has the lowest entropy
#data is a list of dictionaries for which each dictionary attributes(
keys) still need to be split
27 def getLowestEntropy(self ,data):
    temp = data[0]
    returnKey = ""
    entropy = -1.0
    keys = []
32    for k in temp:
        if(k != self.categories[self.numAttributes-1]):
            keys.append(k)
    for key in keys:
37      temp = self.getEntropy(key ,data)
        if(entropy == -1.0):
            entropy = temp
            returnKey = key
        elif(entropy > temp):
42          entropy = temp
            returnKey = key
    return returnKey

```

Figure 20: Program Source Code 6 of 9 [Python]


```

#builds the tree recursively
def build(self, node = None):
    if(self.root is None):
        temp = Node()
        temp.isDecisionNode = False
        temp.isCategoryNode = True
        temp.parent = None
        temp.subset = copy.deepcopy(self.mainData)
        splitCat = self.getLowestEntropy(temp.subset)
        temp.category = splitCat
        childVals = self.getVariableValues(temp.subset, splitCat)
        self.root = temp
        for child in childVals:
            tempChild = None
            tempChild = Node()
            tempChild.category = self.root.category
            tempChild.isCategoryNode = False
            tempChild.isDecisionNode = False
            tempChild.value = child
            tempChild.children = []
            tempChild.parent = self.root
            tempSub = copy.deepcopy(self.root.subset)
            tempSub = self.removeRowFromList(tempSub, temp.category,
tempChild.value)
            tempSub = self.removeKeyFromDicList(tempSub, tempChild.
category)
            tempChild.subset = tempSub
            self.root.children.append(tempChild)
        for child in self.root.children:
            if(child.isDecisionNode != True):
                self.build(child)
    else:
        if(self.isPure(node.subset) == True):
            temp = Node()
            temp.parent = node
            temp.category = node.category
            temp.isDecisionNode = True
            temp.isCategoryNode = False
            temp.subset = copy.deepcopy(node.subset)
            temp.value = self.getDecision(temp.subset)
            node.children.append(temp)
            temp.children = []
            return
        else:
            #node is our parent
            #temp is our new category node
            temp = Node()
            temp.isCategoryNode = True
            temp.isDecisionNode = False
            temp.subset = copy.deepcopy(node.subset)

```

Figure 21: Program Source Code 7 of 9 [Python]

```

2         temp.children = []
        temp.parent = node
        node.children.append(temp)
        temp.category = self.getLowestEntropy(temp.subset)
        catValues = self.getVariableValues(temp.subset, temp.
category)
        #temp is sorted now get children for temp
7         for val in catValues:
            child = Node()
            child.value = val
            child.category = temp.category
            child.isCategoryNode = False
            child.isDecisionNode = False
            child.children = []
            child.parent = temp
            child.subset = self.removeRowFromList(copy.deepcopy(
temp.subset), child.category, val)
            child.subset = self.removeKeyFromDicList(copy.deepcopy
(child.subset), child.category)
            temp.children.append(child)
            for child in temp.children:
                self.build(child)
12
#returns a list of all variable names or different attrbiute names
#within a given data set = data and a category = category
22 def getVariableValues(self, data, category):
    vals = []
    for row in data:
        if (vals == []):
            vals.append(row[category])
27        else:
            new = True
            for x in vals:
                if (x == row[category]):
                    new = False
            if (new == True):
                vals.append(row[category])
32
    return vals
#returns the decision
37 def getDecision(self, data):
    if (self.isPure(data) == False):
        return None
    return data[0][self.categories[self.numAttributes - 1]]
#removes a key from a dictionary list
42 def removeKeyFromDicList(self, data, key):
    for row in data:
        del row[key]
    return data

```

Figure 22: Program Source Code 8 of 9 [Python]

```

1      #removes all rows from a data subset that is not for this category
      value
      #data = list of dictionaries , splitCategory = the category for which we
      are checking a value
      #val = the value within the category for which we only want the rows
      def removeRowFromList(self ,data ,splitCategory , val):
          remove = True
6          while(remove == True):
              remove = False
              for row in data:
                  if(row[splitCategory] != val):
                      data.remove(row)
11                     remove = True
          return data
      #counts nodes and types of nodes
      def countNodes(self):
          if(self.root == None):
16              return 0
          count = 0
          temp = Stack()
          temp.push(self.root)
          while(temp.isEmpty() == False):
21              node = temp.pop()
              count = count +1
              if(node.isCategoryNode == True):
                  self.numCatNodes += 1
              elif(node.isDecisionNode == True):
26                  self.numDecNodes += 1
              else:
                  self.numAttNodes +=1
              for child in node.children:
                  temp.push(child)
31          self.numNodes = count
          return count
      test = ID3()
      print()
      print("The initial entropy values for the given data set:")
36
      for cat in range (0,len(test.categories)-1):
          print("The Entropy for ", end = " ")
          print(test.categories[cat], end = " ")
          print("is ", end = " ")
41          print(test.getEntropy(test.categories[cat], test.mainData))

```

Figure 23: Program Source Code 9 of 9 [Python]

8 Bibliography

- [1] S. Russel and P. Norvig, *Artificial intelligence : A modern approach*, Third. Pearson, 2010.
- [2] V. Lavrenko. (2014). Decision tree, [Online]. Available: https://www.youtube.com/watch?v=eKD5gxPPeY0&list=PLBv09BD7ez_4temBw7vLA19p3tdQH6FY0 (visited on 03/25/2018).
- [3] T. Mitchell. (1997). Machine learning, [Online]. Available: <http://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/mitchell-dectrees.pdf> (visited on 03/25/2018).
- [4] J. R. Quinlan. (1986). Induction of decision trees, [Online]. Available: <https://wwwold.cs.umd.edu/class/fall2009/cmsc828r/PAPERS/%20fulltext%20Quilan%20Ashwin%20Kumar.pdf> (visited on 03/25/2018).