Decision Trees

João Azevedo
up202008367

April 2022

Contents

1. Introduction

Decision trees are tools that helps us classify data based on a set of previously classified data[1].

It is a tree data structure where Nodes represent an attribute, Branches represent a possible value of that attribute and Leafs represent classes. Using this tree to classify new data is as simple as traversing the tree until a leaf node is reached.

Decision trees are very simple and easy to understand since they resemble a human's decision making process[2]. Furthermore they use an open-box model, meaning, using a decision tree we can easily understand which attributes are more important to the outcome, as opposed to black-box models like neural networks where it is difficult to understand how the attributes co-relate.

But decision trees are very susceptible to the quality of the data set. Additionally generating the optimal decision tree is an NP-Complete problem (takes exponential time) which lead most algorithms to follow a greedy approach, the problem is that the greedy approach cannot guarantee that the generated tree will be the optimal decision tree.

2. Decision Tree Algorithms

2.1 Popular Algorithms

There exists many different algorithms to build decision trees. Some of the most popular include the CART (classification and regression trees), ID3 (iterative dichotomiser 3) and C4.5 (ID3's successor).

These 3 algorithms are top-down algorithms [3], meaning, at each level of the tree they choose the attribute that best classifies the current set of data.

Algorithm 1 Top-down decision tree induction algorithms

```
function MAIN(Set,Attributes):
    Node $Root \leftarrow empty$
    if All samples belong to same class then
        $Root \leftarrow$ label class
        return $Root$
    end if
    if $Set$ is empty $or$ $Attributes$ is empty then
        $Root \leftarrow$ most probable class
        return $Root$
    end if
    Attribute $A \leftarrow$ ChoseBestAttribute(Set, Attributes)
    Root's Attribute $\leftarrow A$
    for each value, $V_i$, of A do
        Root's children $\leftarrow Main(Set|A = V_i$, Attributes - A)
    end for
    return $Root$
end function
```

The way each algorithm chooses the best attribute to split the tree, done in the *ChooseBestAttribute* function, is trough metrics: CART uses the Gini Index metric, ID3 uses the Information Gain metric and C4.5 uses the Gain Ration metric. These metric have a basis in information theory.

2.2 ID3 Algorithm

Created by Ross Quinlan in the 80s[4], Iterative Dichotomiser 3 was one of the first decision tree making algorithms it uses Information Gain:

$$InformationGain(S, A) = Entropy(S) - \sum_{V_i \in A} p(V_i).Entropy(S|A = V_i)$$

Where: S is a data set and A is the attribute, V_i is a possible value of A and $p(V_i)$ is the proportion of the number of elements with V_i in S .

To calculate information gain we use entropy which is defined as:

$$Entropy(S) = -\sum_{C_i \in C} p(S|C = C_i).\log_2 p(S|C = C_i)$$

Where: S is a data set, X is the group of all classes in S and $p(x)$ is the proportion of the number of elements from class x in S.

This algorithm has several bottlenecks, mainly: it can't process numerical values and it only works for binary classes.

2.3 CART Algorithm

CART algorithm [5], used for continuous values, this algorithm decides the best attribute to split the tree trough GiniIndex and which first decides the best value to split the attribute trough Gini's Impurity Index :

$$GiniImputrityIndex(S, A, V_i) = 1 - \sum_{C_i \in C} p(S|A = V_i, C = C_i)$$

$$GiniIndex(S) = \sum_{V_i \in A} GiniImpurityIndex(S, A, V_i)$$

Here, A is an attribute, V_i is a value of A, S is the set, C is the group of class and C_i is a value of C, the algorithm then simply chooses the value which wields the highest *GiniImpurityIndex* as a threshold and the attribute which wields the highest *GiniIndex* as the best attribute to split the tree.

Because this algorithm only splits the tree with questions of yes or no the generated decision tree will be a binary tree.

2.4 C4.5 Algorithm

C4.5 was also created by Ross Quinlan[6] to solve ID3's bottlenecks, as such this algorithm is capable of handling numerical values and non-binary classes. Furthermore,it uses information ratio as a metric, opposed to information gain used in ID3.

$$InformationRatio(S, A) = \frac{InformationGain(S, A)}{InformationSplit(S, A)}$$

Where: S is the data set, A is an attribute, $InformationGain$ is the functions used in the ID3 algorithm and $InformationSplit$ is:

$$InformationSplit(S, A) = \sum_{V_i \in A} p(A = V_i).log_2 p(A = V_i)$$

Where: S and A are again the data set and the attribute, respectively, V_i is a possible value of A and $p(V_i)$ is the proportion of number of elements with V_i in S.

3. C4.5 Implementation

Note: See appendix to know the purpose of the used data types as well as the definition, input, output, complexity and brief description of the used functions.

Firstly, the program will take 2 files as input, the first being the classified data and the second the test data.

3.1 Read and prepare classified data

It then begins by reading the classified data from the first file using $readClassified$ function. In this function the global variables $Dictionary dictionary$ and $DataSet classified$ will be populated.

After all data is read from the first file the function $C45$ is called. This function prepares data and inputs for the decision tree building, this includes using *binarySplit*, generating the *validEntries Set* and *validAttributes Set* as well as using the *printData* function to print the data table.

binarySplit is a function that transforms numeric attributes into binary, yes or no, attributes, for this the algorithm first sorts all the values of the attribute. Using the the sorted array the program creates a table, this table has the different values of classes as columns (plus 1 for total) and each value as a line, in each line is stored the number of occurrences of the classes until that value(including the value). The using the table we calculate the $informationGain$ wielded if the value was the *threshold*. Then we simply set as *threshold* the value that wielded the highest $informationGain$.

3.2 Building the tree

Now, with all data prepared, function *buildTree* will be called, this function does what was described in Algorithm (1), for that we use 2 functions: *checkClasses* because this functions returns the number of occurrences of each class and the total entries we can use it to see if only one class appears (occurrences of class will be = to total entries) and , in case *validEntries* is empty *or* *validAttributes* is empty which is the most probable class; and *chooseBestAttribute* to choose the attribute that better divides the data.

chooseBestAttribute is where all calculations occur. To set up for those calculations the programs iterates trough the *classified DataSet* and builds a table for each of the arguments. This table has one line for each value of the attribute it is associated with and a column for each class value existent in *classified*, plus a line and column for totals. Then we calculate *globalEntropy* (only once since it does not change from attribute to attribute), *weightedEntropy* (the entropy for each value of the attribute multiplied by the probability of the value), *informationGain*, *informationSplit* and finally *informationRatio*. Then we simply return the index of the attribute that wields the highest *informationRatio*.

Then for each value of the chosen attribute we create a *newEntries Set*, copy *validEntries* to it but setting the *ith* boolean to *false* if the entry does not have that value in the chosen attribute column. Then we recursively call *buildTree* using the *newEntries* array.

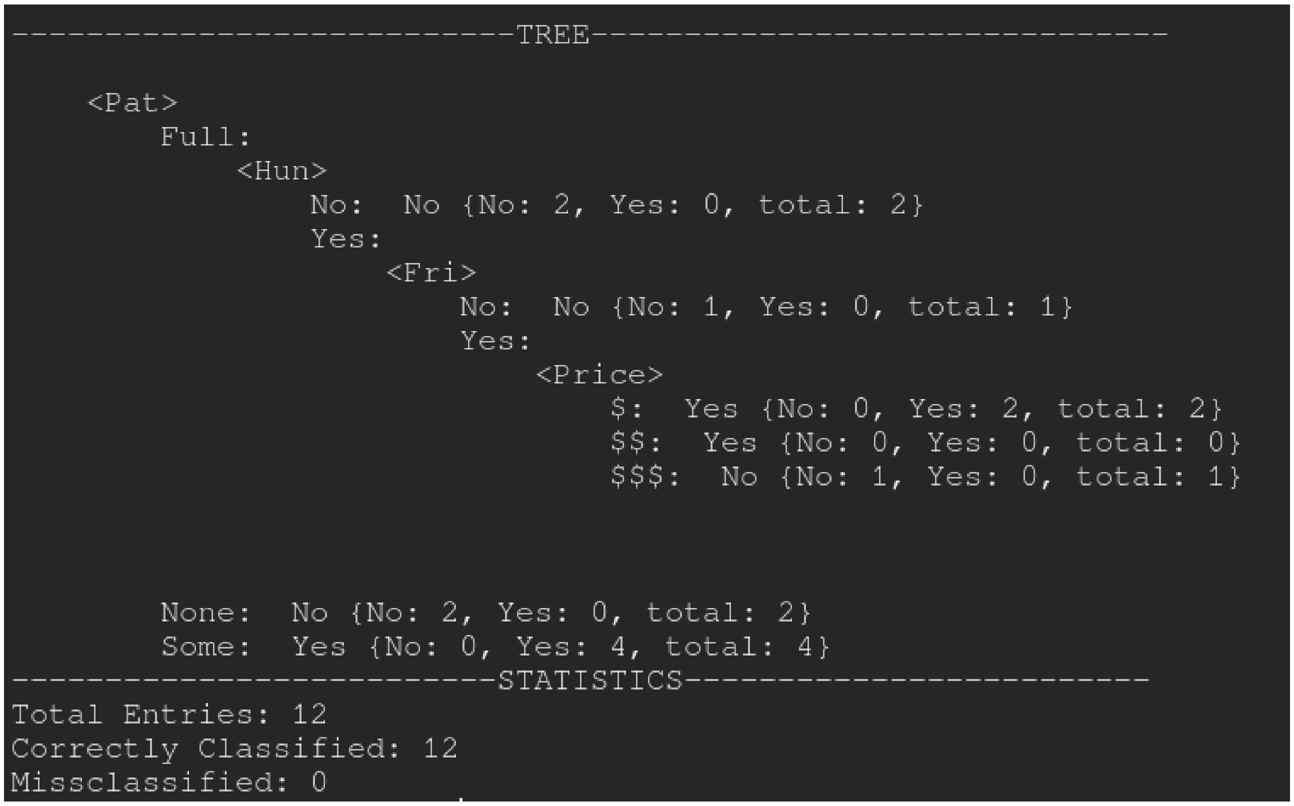
After finishing building the tree, the program prints all the *Node* in the tree using *printTree*.

3.3 Classifying test data

Now that the tree is finished and printed to the screen the program will use *readToClassify* to read the second .csv file. This second file is expected to have all the attributes (including the id attribute) but a black class, meaning each line is expected to end with a comma.

Upon finishing reading the table of the second file the data is classified using the *classify* function and the result will be displayed in the screen, the *ith* class is the classification of the *ith* line.

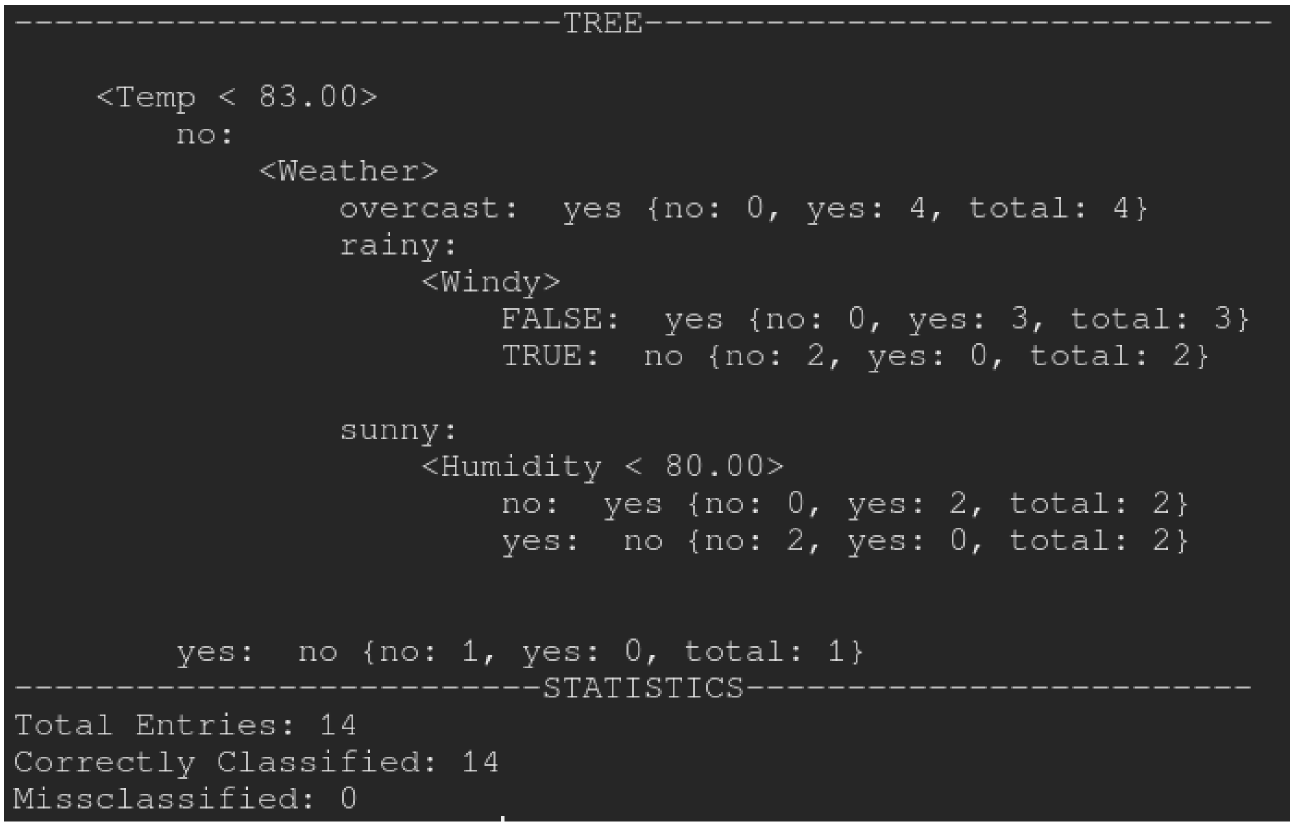
4. Results



```
---------------------------TREE-------------------------------

    <Pat>
        Full:
            <Hun>
                No:  No {No: 2, Yes: 0, total: 2}
                Yes:
                    <Fri>
                        No:  No {No: 1, Yes: 0, total: 1}
                        Yes:
                            <Price>
                                $:  Yes {No: 0, Yes: 2, total: 2}
                                $$:  Yes {No: 0, Yes: 0, total: 0}
                                $$$:  No {No: 1, Yes: 0, total: 1}



        None:  No {No: 2, Yes: 0, total: 2}
        Some:  Yes {No: 0, Yes: 4, total: 4}
--------------------------STATISTICS--------------------------
Total Entries: 12
Correctly Classified: 12
Missclassified: 0
```

Figure 4.1: The Tree generated by the restaurant data set

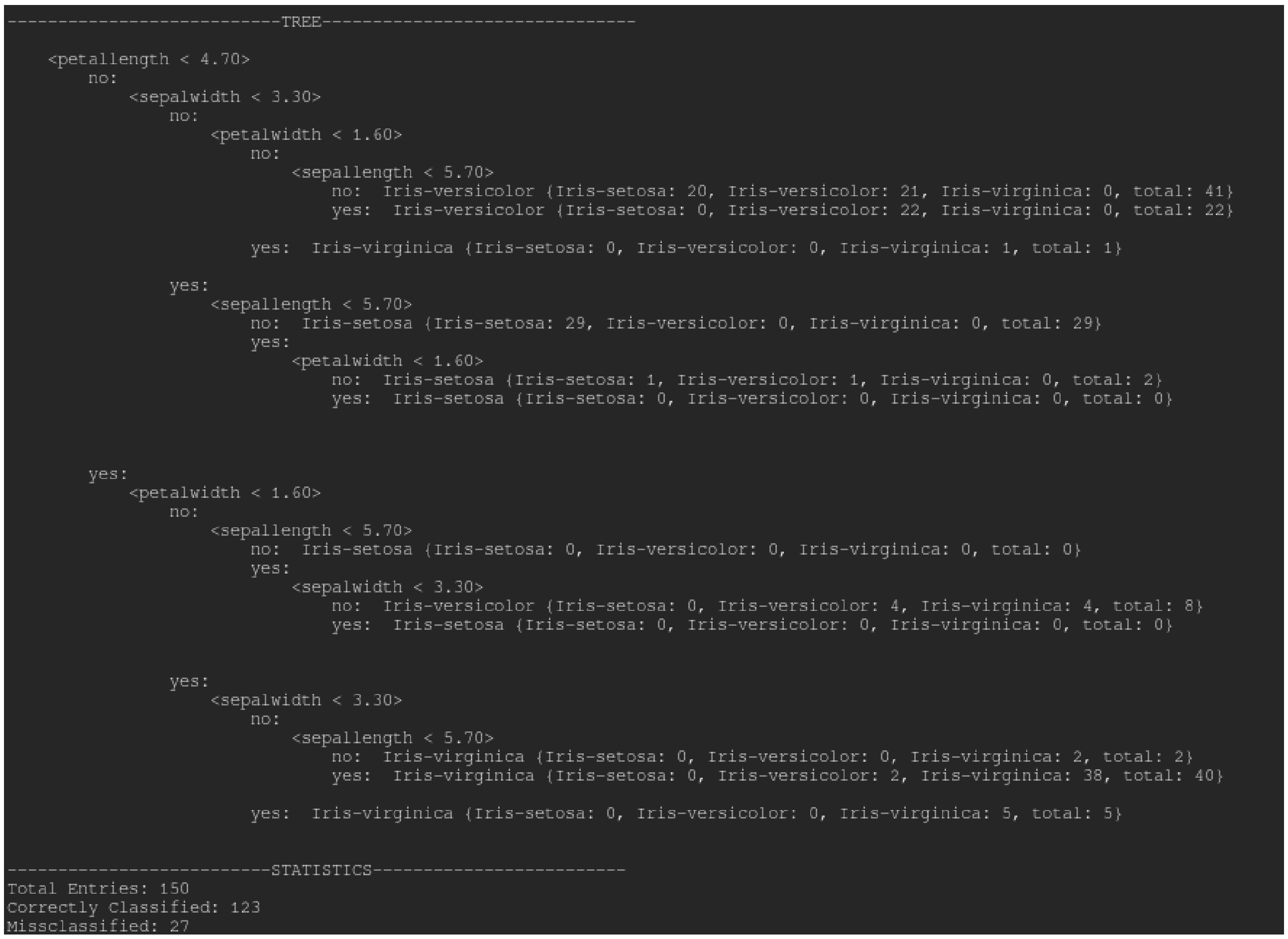
This data set has 10 attributes, all discrete. The algorithm generated a tree composed of only 4 Nodes, and classifying correctly all the input data.



```
---------------------------TREE-------------------------------

    <Temp < 83.00>
        no:
            <Weather>
                overcast:  yes {no: 0, yes: 4, total: 4}
                rainy:
                    <Windy>
                        FALSE:  yes {no: 0, yes: 3, total: 3}
                        TRUE:  no {no: 2, yes: 0, total: 2}

                sunny:
                    <Humidity < 80.00>
                        no:  yes {no: 0, yes: 2, total: 2}
                        yes:  no {no: 2, yes: 0, total: 2}


        yes:  no {no: 1, yes: 0, total: 1}
--------------------------STATISTICS-------------------------
Total Entries: 14
Correctly Classified: 14
Missclassified: 0
```

Figure 4.2: The Tree generated by the weather data set

This data set has 4 attributes with 2 of them being numerical. In the tree we can see that the threshold of the temperature attribute is 83.0 and the threshold of the humidity attribute is 80.0. This data set was also perfectly classified.



```
--------------------------TREE------------------------------

    <petallength < 4.70>
        no:
            <sepalwidth < 3.30>
                no:
                    <petalwidth < 1.60>
                        no:
                            <sepallength < 5.70>
                                no:  Iris-versicolor {Iris-setosa: 20, Iris-versicolor: 21, Iris-virginica: 0, total: 41}
                                yes:  Iris-versicolor {Iris-setosa: 0, Iris-versicolor: 22, Iris-virginica: 0, total: 22}

                        yes:  Iris-virginica {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 1, total: 1}

                yes:
                    <sepallength < 5.70>
                        no:  Iris-setosa {Iris-setosa: 29, Iris-versicolor: 0, Iris-virginica: 0, total: 29}
                        yes:
                            <petalwidth < 1.60>
                                no:  Iris-setosa {Iris-setosa: 1, Iris-versicolor: 1, Iris-virginica: 0, total: 2}
                                yes:  Iris-setosa {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 0, total: 0}



        yes:
            <petalwidth < 1.60>
                no:
                    <sepallength < 5.70>
                        no:  Iris-setosa {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 0, total: 0}
                        yes:
                            <sepalwidth < 3.30>
                                no:  Iris-versicolor {Iris-setosa: 0, Iris-versicolor: 4, Iris-virginica: 4, total: 8}
                                yes:  Iris-setosa {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 0, total: 0}


                yes:
                    <sepalwidth < 3.30>
                        no:
                            <sepallength < 5.70>
                                no:  Iris-virginica {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 2, total: 2}
                                yes:  Iris-virginica {Iris-setosa: 0, Iris-versicolor: 2, Iris-virginica: 38, total: 40}

                        yes:  Iris-virginica {Iris-setosa: 0, Iris-versicolor: 0, Iris-virginica: 5, total: 5}


-------------------------STATISTICS-------------------------
Total Entries: 150
Correctly Classified: 123
Missclassified: 27
```

Figure 4.3: The Tree generated by the iris data set

This data set has 3 attributes, all of them numerical. Here we can see that 27 entries were miss classified, 18%, this happens because the attributes are all numerical so the thresholds were used to classify the most entries possible leaving some entries to be miss classified. As such the bulk of the miss classified data are the entries that were given a value of "no" for all the attributes, 20 entries.

5. Conclusions and Comments

For the classification of data decision trees are very powerful algorithms.

Although it has some limitations as being heavily depend on good data, for example: a value that was not previously seen appears in the test data all we can do is assign the class that appears the most in that node, and only dividing numerical values in a binary class, this ca be seen in data set 3, iris, where all entries that are classified with no in the 3 attributes have almost 50% chance of not being correct.

The algorithm allow us to know exactly what the generated tree is and how it is classifying data. Furthermore it also generates very small trees, although there is no way of knowing if the generated tree is the smallest since it is a greedy algorithm.

With these results it also becomes clear why C4.5 is a widely used algorithm for decision tree induction, with a decent complexity it tries to build the smallest tree by not using $InformationGain$ but using $InformationGain$ which leads to attributes that have less branching factor being choosen before those with high branching factor leading to a smaller tree.

Bibliography

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 2002. URL: `http://aima.cs.berkeley.edu`.

[2] Sotiris B Kotsiantis. "Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (2013), pp. 261–283.

[3] L. Rokach and O. Maimon. "Top-down induction of decision trees classifiers - a survey". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 35.4 (2005), pp. 476–487. DOI: `10.1109/TSMCC.2004.843247`.

[4] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[5] Roman Timofeev. "Classification and regression trees (CART) theory and applications". In: *Humboldt University, Berlin* 54 (2004).

[6] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

A. API

This is a place where all functions will be enumerated and documented. For each data type there should be: the members of the data type and a little description. For each function there should be: the function definition, the parameters, the return value, mini description and its complexity.

A.1 Base File

A.1.1 Basic Data Types

The *String* wraps an array of characters of arbitrary size (defined as 32 but can be extended), has a custom comparison function. Used to save the names of attributes used in the data set.

The *Attribute* type, used to represent all possible values of an attribute(*String*) and associate them to an unique *integer* (uses a *map*), also has s boolean which is *true* if the attribute is numerical and a double, the threshold for numerical values, this means that values greater than the threshold will be classified as *true* and values less than or equal to it will be $false$.

The *Dictionary* type, a *vector*(*array* that can be resized) to group all of the different *Attributes*.

The *Entry* type, a *vector* of *integer*, each of these *integers* is the unique *integer* associated to a value of an attribute. The attribute to which it is referring has the same column index in the *vector* and in the *Dictionary* (the first *integer* in an *Entry* is a value of the *Attribute* in the first column of the *Dictionary*).

The *DataSet* type, a *vector* of *Entry*.

The $Node$ type, encapsulating pointers to its children (also $Nodes$), the index of attribute it is referring to, a *boolean* that is *true* if the $Node$ is a Leaf, as well as a *vector* of integers which are the counters of each class .

Finally the $DecisionTree$ type, which is the root $Node$ of the decision tree.

The *Set* type, a *vector* of *booleans* and an *integer*, this type is used to represent a *Set* of entities, meaning if the *ith* position of the *vector* is *true* then the *ith Entry* or *Attribute* is part of the set, the *integer* is simply the reference for how many entities are part of the *Set*.

A.1.2 Global Variables

Using the data types defined before there are 2 global (can be accessed anywhere) objects, the first being a dictionary and the second being the first input data set (classified data) named *dictionary* and $classified$ respectively

These objects are global because they are used in almost every function of the program.

A.1.3 Useful methods using the data types

- $dictionary[i].size$ wields the number of different values that the *Attribute* i has.
- $dictionary.size()$ wields the number of different *Attributes* that will be considered (including the class attribute).
- $dictionary.size() - 1$ is the index of the class attribute in the dictionary
- $dictionary[dictionary.size() - 1].size$ is the number of different values for the class attribute

A.2 Nomenclature for complexity:

N_1, number of *Entry* in the first file (the one that is already classified)
N_2, number of *Entry* in the second file (the one that isn't classified)
A, number of *Attribute* (a.k.a. columns) of the data
A_i, number of values of *Attribute* i
A_v, average quantity of values

A.3 Decision Tree File

A.3.1 C4.5 Tree Builder

$Node\ C45()$

- initializes everything needed to build the tree (call $buildTree$)
- returns the $Node$ at the root of the decision Tree
- $O((A_v)^A)$

$void\ binarySplit()$

- Circles trough the attributes and for the attributes that are numerical will, trough information gain, decide which value will be set as a threshold, after deciding the threshold it will change the values in $classified$ table to yes or no as explained above.

- $O(N_1 * A)$

$Node\ buildTree(Set\ validEntries,\ Set\ validAttributes)$

- Takes 2 *Sets* the first representing the entries that are to be evaluated and the second representing the unused attributes
- Recursively builds the decision tree
- Returns a Node that is the best (according to the metrics) attribute for the Entries and Attributes given
- $O((A_v)^A)$

int chooseBestAttribute$($*Set validEntries*$,$ *Set validAttributes*$)$

- As input it takes 2 *Sets*, the first representing the *Entries* of the data set that are valid and the second representing the unused *Attributes*.
- Chooses the *Attribute* from the *validAttribute Set*, that best divides the *Entries* given in the first *Set*.
- It returns an *integer* that is the index of the chosen attribute.
- $O(N_1)$

$int\ classify(Node\ root,\ Entry\ a)$

- takes an *Entry* object as input
- Classify it using the built decision tree
- returns the index of the value which a was classified with
- $O(A)$

A.3.2 Helper Functions

double inline xlnx$($*double* x, *double* $y)$

- takes 2 doubles as input
- returns $x/y * log_2(x/y)$
- note that if $x = 0$ *or* $y = 0$ the function returns 0

vector¡int¿ checkClasses$($*Set validEntries*$)$

- takes a *Set* of valid entries as input
- will iterate trough the set and count the number of different classes
- outputs a vector with the counters of each class and, at the end, the total of entries iside the set

A.4 I/O Handler file

$void\ readClassified(FILE * f)$

- takes a file(in csv format) as an input
- populates the $dictionary$ and $classified$ objects
- $O(N_1 * A * log(A_i))$ complexity with N as the number of $Entry$ in $classified$, and A the number of $Attributes(dictionary.size())$

$dataSet\ readToClassify(FILE * f)$

- takes another file(also csv format) as input, this time with no class attribute
- for each $Entry$ read from file classify it using $classifyNew$
- $O(N_2 * A * log(A_i)) * O(A)$

$void\ printTree(Node\ n,\ int\ tabs = 0)$

- takes the number of tabs(indentation needed to put before printing the node, default = 0)
- prints the content of the $Node$ to standard output

$void\ printData()$

- prints, to standard output the data inside $classified$