Assignment 3 - Decision Trees

Jordi Garcia & Miquel Roset

May 26, 2022

1 Introduction

This report exposes and analyzes the work done in the third assignment of the Artificial Intelligence laboratory.

The structure of this document consist of an introduction on decision trees and what they are used for. Following that, the report studies some of the most popular algorithms for decision tree induction and exposes some insights about them. Implementation wise, this document goes over the technical details used to implement the ID3 algorithm. The report includes the generated trees from each data set provided and it ends with some final comments and conclusions.

1.1 What is a decision tree?

A **decision tree** represents a function that takes as input a vector of attribute values and returns a "decision" —a single output value [1, p. 698].
The input and output values can be discrete or continuous. A decision tree reaches its decision by performing a sequence of tests. Each internal node in the tree corresponds to a test of the value of one of the input attributes. [1, p. 698]

1.2 What is it for?

The aim of the decision tree is to learn the definition of the **goal predicate** [1, p. 698]. In other words, the decision tree will work as a classification or regression model that has learned some behavior of a single attribute based on all the others. This allows to predict some properties of that attribute or even the exact value, whenever it is unknown.

2 Algorithms for decision tree induction

2.1 Popular algorithms

Some popular algorithms, each of them for their own reasons, might well be **ID3**, **C4.5**, **C5.0**, **CART**, **CHAID**, **MARS** and **QUEST**.

2.2 Main differences

Among this decision tree induction algorithms we can identify four main differences:

1) **Model type:** Algorithms can build classification models used for discrete variables (e.g: a class label), regression models used for continuous variables (e.g: score or age) and there are some algorithms that are capable of building both.

2) **Splitting criteria:** Algorithms differ in how variance is measured for a data set. For instance, ID3 finds splitting points using the Information Gain considering the entropy and arithmetic means whereas the QUEST algorithms goes as far as using Chi Square tests.

3) **Over-fitting:** There are different ways to eliminate the over-fitting an thus perform better on unseen data.

4) **Incomplete data:** Some algorithms can handle incomplete data whereas others simply cannot.

2.3 Choosing attributes

As stated in the previous section, the splitting criteria is one of the aspects that identify each algorithm. The algorithm implemented for this assignment is ID3 which is well fitted to use the Information Gain metric. Within the context of the laboratory, the process of choosing an attribute among all the others has used the Information Gain metric exclusively.
Its worth mentioning that even though it is the only metric used to choose attributes, it is not possible to obtain the Information Gain of a continuous attribute right away. This is due to the fact that a continuous attribute would divide the data set into an infinite number of subgroups if treated like a discrete one. For this reason, another metric that has been mandatory to consider when considering continuous attributes is the splitting point.
The splitting point for continuous attributes is a reference value that should divide all the attribute data into two sets: the ones above or equal, and the ones below the splitting point. To find out the reference value which best splits the data, first are determined the candidates splitting points and, then, chosen the one which maximize the information gain. The candidates are those points between two attribute sorted values which its classification differs.

2.4 ID3 algorithm

ID3 is an algorithm that builds a classification model by inducing a decision tree from a given data set S.
The algorithm starts with the original data set S as the root node. From there, it looks for partitioning S using the attribute for which the resulting entropy after partitioning is minimized. Another way to express this procedure is that it looks for the attribute that maximizes the Information Gain. Once it has identified the attribute, the algorithm assigns it to the current node and uses it to split the data set

into subgroups. From here, the algorithm uses the remaining attributes to recurse over the newly obtained subsets. The algorithm is able to stop recursion whenever:

- Every element of the subset belongs to the same class.
- There are no more attributes left, but the attributes do not belong to the same class. In this case, the node is labeled with the most common class of the subset.
- There are no examples in the subset. In this case, the node is labeled with the most common class of the examples from the parent node.

3 Implementation

3.1 Language

The language chosen for the implementation is *C++*, on its version 11.2.0.

3.2 Code organization

src/dataset.cc: This file contains the implementation of the *DataSet* class.

inc/dataset.hh: This file contains the definition of the *DataSet* class.

src/decisiontreenode.cc: This file contains the implementation of the *DecisionTreeNode* class.

inc/decisiontreenode.hh: TThis file contains the definition of the *DecisionTreeNode* class.

src/id3.cc: This file contains the implementation of the *ID3* algorithm function. This file uses the *DataSet* an *DecisionTreeNode* classes.

inc/id3.hh: This file contains the definition of the *ID3* algorithm function.

inc/cxxopts.hh: External library that contains useful functions and operators in order to parse command line arguments.

src/main.cc: This file contains the *TBD* structure with some methods that

3.3 Data structures

3.3.1 *UMSI* map

The *UMSI* map is one of the most used data structure during the assignment. It is essentially an *unordered_map<string, int>*, which is used to store a counter for labels. The labels typically are the possible attribute or classification values, and the counter, the occurrences of these labels in the data set.

3.3.2 *DataSet* class

The *DataSet* class represents the data set information and its operations. This class consists on the following properties and functions:

string *filename*: An string containing the path to the training data set, as a comma-separated values format.

vector<vector<string» *content*: A matrix that holds the information of the data set file.

vector<unordered_map<string,pair<int,*UMSI*»> *attributes*: A vector of maps that contains, for each key label, corresponding to possible values of the attribute, a pair of an *int* and a *UMSI* structure which contains the number of occurrences of that attribute value and its classification label counters, respectively.

vector<bool> *attribute_continuous*: A vector that contains for each attribute if it is continuous or not.

vector<string> *attribute_headers*: A Vector containing the names of the attributes.

***UMSI classes*:** A *UMSI* structure to save a counter for each classification label.

int *class_index*: The column index of the *content* matrix corresponding to classifications.

float *entropy(int count, UMSI& values)*: Function that returns the entropy value given a total number of examples (*count*), and a *UMSI* structures containing classification counters corresponding to an attribute value.

bool *is_continous(string value)*: Function that returns true if the string given correspond to a continuous value or not. Namely, if the string *value* correspond to an integer or float representation.

string *get_class(int row)*: returns the classification label given the index of an example in the *content* matrix.

bool *classEq(vector<int> &rows)*: returns true when the classification labels corresponding to the example rows defined by the *rows* vector are all the same.

pair<string, int> *plurality_value(vector<int> &rows)*: returns a pair containing the most common classification label and the number of occurrences given a vector of rows corresponding to examples in the *content* matrix.

void *load(string filename)*: Parses the *filename* file and fills all the corresponding properties of the class.

float *importance_continuous(...)*: returns the importance of a given attribute and a vector of rows, as well as the splitting point and the subsets generated to compute it.

float *importance_discrete(...)*: returns the importance of a given attribute and a vector of rows, as well as the subsets generated to compute it.

3.3.3 *DecisionTreeNode* class

The DecisionTreeNode class represents a node of the induced decision tree. This class consists on the following properties and functions:

int *attribute*: The index corresponding to the attribute tested on the current node.

int *count*: The count of examples remaining when the current node is a leaf.

float *split_point*: The splitting point value when the attribute is continuous.

bool *continuous*: A boolean value specifying if the current node corresponds to a continuous attribute.

string *classification*: The classification label when the current node is a leaf.

vector<pair<string,unique_ptr<DecisionTreeNode»> *children*: A vector of pairs of an string, corresponding to the attribute value that match the branch, and a pointer to the subsequent tree node corresponding to that value.

bool *is_leaf()*: returns true if the current node is a leaf.

void *set_classification(string label, int count)*: Mark the current node as a leaf, setting the classification label and the counter.

void *set_splitting_point(float sp)*: Mark the current node as continuous attribute node, setting the splitting point used to discretize the data.

void *add_branch(string label, unique_ptr<DecisionTreeNode> subtree)*: Add a new pair to the *children* vector.

void *print(vector<string>& attribute_labels, int depth)*: Prints the decision tree as defined in the assignment.

void *decide(vector<string>& row)*: Given a non-classified example, performs a lookup in the decision tree to assign a classification label.

4 Results

This assignment has provided four different data sets. One of them with only discrete attributes and the two remaining mixing discrete and continuous attributes.

4.1 Restaurant.csv

This data set is the simplest one because it only contains 12 examples and discrete attributes only. The tree induced using the implemented ID3 is shown in Listing 1.

```
<Pat>
    None: No (2)
    Full:
        <Hun>
            No: No (2)
            Yes:
                <Type>
                    Italian : No (1)
                    Thai:
                        <Fri>
                            Yes: Yes (1)
                            No: No (1)
                    Burger: Yes (1)
                    French: No (0)
     Some: Yes (4)
```

Listing 1: Tecision tree for Restaurant data set.

It is worth noticing that the counter of the *French* leaf from Listing 1 is zero. This is due to the fact that when building the node that classifies the attribute *Type*, there are no remaining examples with value *French*. The decision of the tree at that leaf is determined with the plurality value of the parent examples, which are 1 *Italian*, 2 *Thai* and 1 *Burger*. The result, labels the *French* leaf with examples that will also be used to label other leafs. The fact that a leaf counter is zero can be used to determine that the tree might be suffering from *overfitting* because it has no data when *Type* equal to *French* at that point of the tree.

4.2 Weather

The Weather data set is still quite simple with only 14 examples to learn from. It has two continuous attributes which are *humidity* and *temperature*. The induced tree is shown in Listing 2.

```
<Weather>
    rainy:
        <Windy>
            TRUE: no (2)
            FALSE: yes (3)
    overcast: yes (4)
    sunny:
        <Humidity>
            <77.500000: yes (2)
            >=77.500000: no (3)
```

Listing 2: Tecision tree for Weather data set.

4.3 Iris

The Iris data set is the larger used with 150 examples to learn from. Except from the class, all the attributes are continuous which makes the induction of the tree significantly slower. The decision tree induced is shown in Listing 3.

```
<petallength>
    <2.450000: Iris-setosa (50)
    >=2.450000:
        <petalwidth>
            <1.750000:
                <sepallength>
                    <7.100000:
                        <sepalwidth>
                            <2.850000: Iris-versicolor (27)
                            >=2.850000: Iris-versicolor (22)
                    >=7.100000: Iris-virginica (1)
            >=1.750000:
                <sepalwidth>
                    <3.150000: Iris-virginica (32)
                    >=3.150000:
                        <sepallength>
                            <6.050000: Iris-versicolor (1)
                            >=6.050000: Iris-virginica (13)
```

Listing 3: Tecision tree for Iris data set.

5 Final comments and conclusions

a. A leaf node with a counter of zero (e.g: the *French* leaf node of the Listing 1) indicates that the tree has no data to learn from at that stage. This indicates that unseen data might be poorly predicted at that leaf. This observation could be some form of *overfitting*.

b. Regarding the data set from Section 4.3, we've seen that some examples used for training do not get properly classified with the induced tree. This means that the hypothesis space is not able to represent the pattern of the data. For this data set we should try a more complex model thus we can say that the one used potentially suffers from *underfitting*.

c. The strategy used to discretize continuous data is one of the most expensive parts of the *ID3* algorithm. Moreover, we consider that the splitting strategy, described in Section 2.3, is not well fitted for the data sets used. There are other approaches that should be explored in order to provide a much finer granularity for the divisions, such as splitting into *n* subgroups instead of only two.

References

[1] S. Russell et al. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. ISBN: 9780136042594. URL: https://books.google.pt/books?id=8jZBksh-bUMC.