PREVENTING ROYA IN COFFEE PLANTS USING DECISION TREES

Juan Sebastián Diaz Osorio
Universidad EAFIT
Colombia
jsdiazo@eafit.edu.co

Liz Oriana Rodrigues Cruz
Universidad EAFIT
Colombia
lorodriguc@eafit.edu.co

Mauricio Toro
Universidad EAFIT
Colombia
mtorobe@eafit.edu.co

SUMMARY

The coffee leaf rust it is a fungus that implants in the leafs of the coffee plants searching for survival, as they do that they became one of the most dangerous sicknesses that a plant can get, as it comes with a price to pay because a large amount of coffee powder does not have a way of recovery, as a consequence after one these plantations become sick, it has to extracted and fired up immediately and by that means the lost of the product, the coffee.

Machine Learning [7] would be an amazing support tool for this situation. Imagine a computer that it studies each second the crop ground characteristics and alert about coffee leaf rust occurrence. Thus, the plague start point would be known and could be eliminated.

Based on the document you would find the most efficient solution to define if a coffee plant poses one of this disease based on the statements. Detect the royan in these plants is a risky and delicate process because it is necessary to have immediate answers and by this we could prevent other plants of being infected and also it can help to prevent the use of these for consuming just because it can cause such health troubles, the solution is by using algorithms C4.5 which helps by using all the data collected to classify and define if the plant has or not the infection.

KEYWORDS

Coffee leaf rust, Coffee Plants, Machine Learning, Dangerous Sicknesses, The Crop Ground, Parasites, Transgenic Coffee, Collecting Data, Algorithm, Computer Solutions, Learning Algorithms, Decision Tree, Data Attribute, Dataset, Induction Decision Trees, Artificial Intelligence, Backtracking, Information Gain, Recursion, CART, Decision Tree Classifier, Supervised Learning In Quest, Nodes, Tree of Data class, Data Structure, Big-O Complexity.

ACM CLASSIFICATION

Computing Classification System → Poly-Hierarchical → Semantic Web → Standard Classification → Evolves in the Future → Search Interface → ACM Digital Library → ACM Press → Quick Content → Related Literature → Online Resources → LaTeX → CCS → ACM → Full-Text Collection → Guide to Computing Literature → Connections → Hardware → Software → Theory of Computation → Applied Computing → Biographies → Data Management Systems → Database Design and Models → Networks → Public Internet → Classification Tree.

1. INTRODUCTION

The management and crop care is a hard job if you don't have correct tools, because diseases and parasites are avoided only when they are prevented.

The coffee, Colombia's main export product, is affected by a fungus called "coffee leaf rust", which changes his leaves' color into a yellow tone until withering and the leaves fall, jamming his maturation and production process.

Even though laboratory tests have originated a transgenic coffee that is invulnerable to coffee leaf rust [2], the replacement for this new crop could bring changes in flavor or care topics. In this text, we analyze the possibility to study the ground by collecting data and prevent coffee leaf rust occurrence in coffee plants using some computer solutions.

2. PROBLEM

We look for preventing coffee leaf rust occurrence in the coffee plants to avoid loss, through machine learning algorithms that analyze terrain variables and foretell when and where is safe to farm.

3. RELATED WORKS

A decision tree is a machine learning method that classifies new data. It has a tree-like structure made of nodes (linked with other through branches) and logic test branches (which allows going from a node to another). New data cross through this tree and when they finish it, they will have a new label.

An algorithm builds the tree from a dataset (called training data). More training data implies more accuracy in the decisions.

Now, we briefly show algorithms that create decision trees:

3.1 ID3 [1]

J. Ross Quilan, in 1979, created the ID3 algorithm (Induction Decision Trees). It uses artificial intelligence encompassing the search for a thesis or rules by means of a set of examples. In addition, it is a constructive algorithm to obtain decision trees based on the CLS algorithms.

It highlights the application and realization of decision trees from top to bottom, directly, without having to make use of Backtracking, however, this uses recursion and is based on the examples provided. For that, one must have basic knowledge of what is and how the concept of Information Gain is measured, which consists of identifying the most useful attribute for the process to be carried out.

To solve a problem through ID3 algorithms we must take into account elementary details such as the inputs, which consists of a list of examples with data or values, each stored with its attribute, the output, which consists of the tree of final decision that will have separated the examples with their respective classes, which can be predefined or discrete.

The decision tree is traversed from the root and a tour is made throughout its structure, that is, root and nodes based on the value of the attribute in classification until it reaches a terminal node.

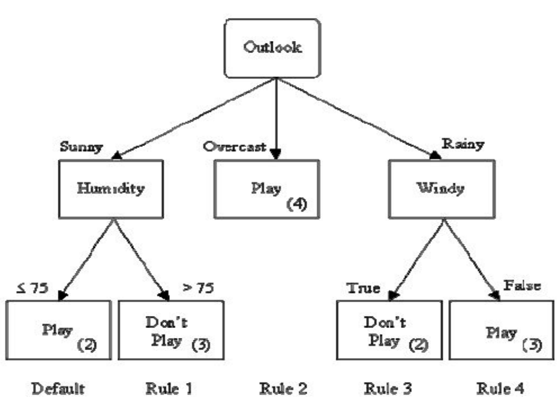
One of which of the trees with this type of algorithm is that it can adapt to certain types of problems that can be had.

3.2 C4.5 [4]

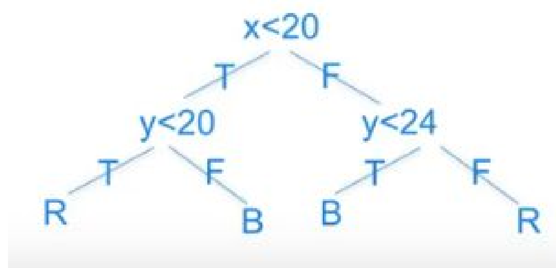
C4.5 algorithm is an extension of ID3, but this uses recursion to generate the decision tree from training data. It considers each data attribute to split the dataset into new groups, based on the highest information gain (this is a data mining term. It means that a variable can be used as a classifier thanks to great accuracy).

This algorithm works for both discrete and continuous attributes, doing either a big data test (with n possible values) or binary test (two values only) to find the information gain. The system analyzes all data to decide which is the logic question more efficient and accuracy to do.

Some advantages over ID3 are:
- It works even if there is missing information
- It has a good computational efficiency
- It generates smaller decision trees.
- It includes a post-pruning process, where the tree is evaluated again to improve performance.



A C4.5 algorithm example is at scottjulian's repository [7]. It is implemented in Java and a great idea about how solve our problem.



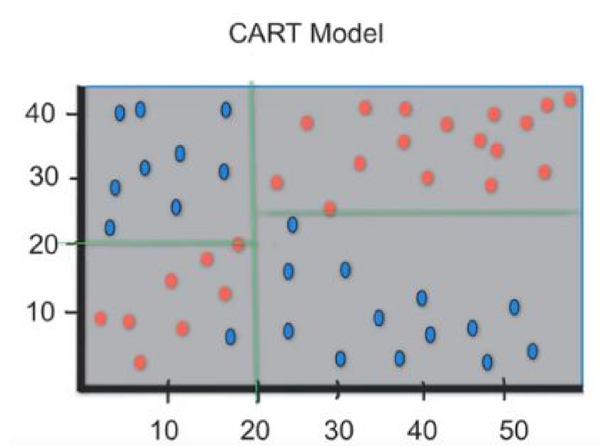
3.3 CART [3]

When a dataset can be classified with two labels only, (it means either “A group” and “B group”, or “Full” and “Partial”...), a good idea to create a decision tree model is the CART algorithm.

Let x and y axis be two numerical categories of a dataset in the next picture.

Each data in the dataset can be positioned in the diagram and they can be split into subgroups. Each subgroup will have a dominating label and it means new data has a high probability to have that label.

CART algorithm generates a decision tree like this:



So, new entry data can be positioned and classified in the diagram according to values with good accuracy.

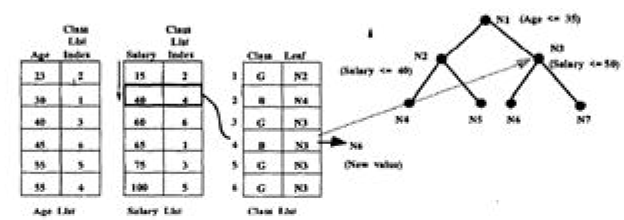
3.4 SLIQ [5]

SLIQ (Supervised Learning In Quest) is a decision tree classifier that can handle both numerical and categorical attributes. It stands out because it can handle a big amount of training data (unlike other algorithms) thanks to its pre-sort method, acquiring awesome accuracy and efficiency.

First, it creates a sorted list for each numerical and categorical attribute, including a list for the classes and tree nodes (class list).

Then, it builds nodes and links it according to previous lists, creating and updating nodes for the class list. Next, the algorithm deletes cloned nodes.

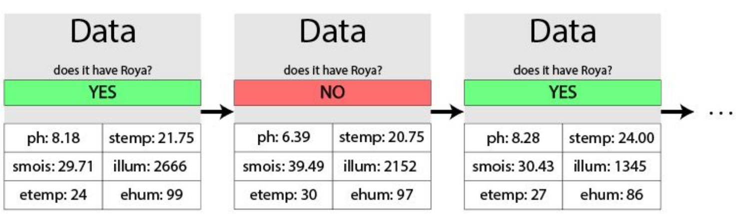
In the tree pruning, the tree is codified while some nodes are deleted. At this point, the class list will have each class with its respective node and new data must cross the tree, reach a node, and compare itself with the class list to be classified.



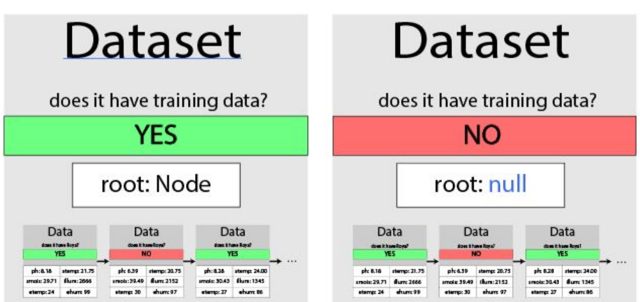
In performance tests, SLIQ shows to be faster than IND-CART and generates smaller trees than IND-C4.

4. DATA STRUCTURE

According to the fastest search and insert necessity, we decided to use an LinkedList, filled with Data class elements which refers to each data in a dataset:

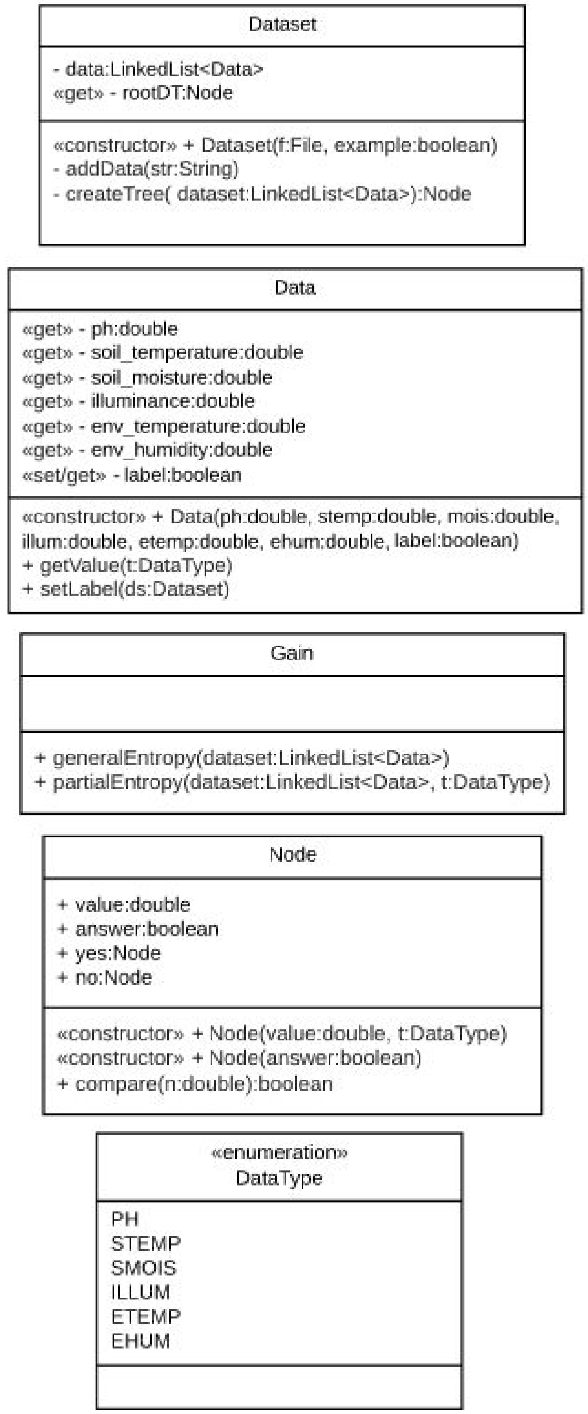


It is a Dataset. Any Dataset can have a comparison tree whose existence depends on if it is for training data:



And this is our data structure proposal. Name is not defined yet.

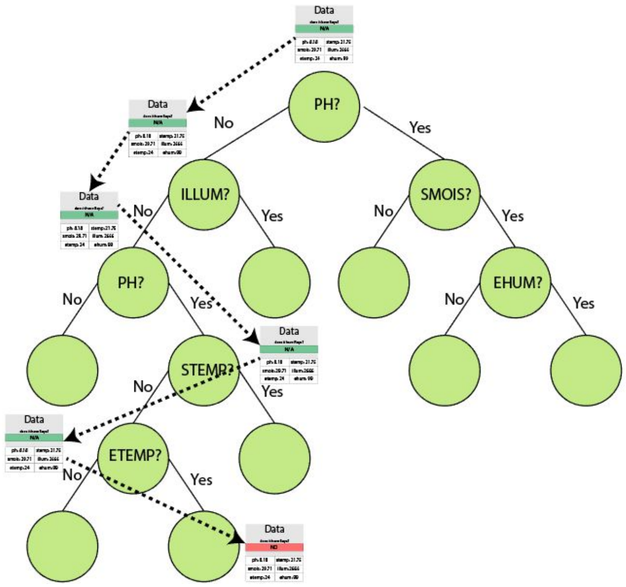
Next, our implementation in Java explained via UML Class Diagrams:



The Java implementation is explained through Javadoc in Spanish on this link: https://bit.ly/2orTIzb

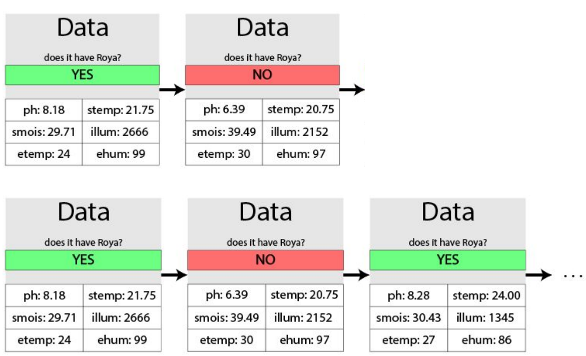
4.1 OPERATIONS

setLabel (Data): With a given Training Dataset, a data with **setLabel** method gets the comparison tree and classifies itself, changing its label.

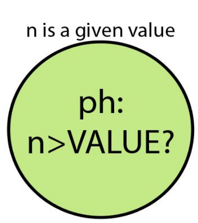


createTree (Dataset): With a given LinkedList of Data, a dataset with **createTree** method defines a node that separates the dataset according to information gain of best separation variable. It is a recursive method which allows create sons of the node, also. This method creates a binary comparison tree like previous picture.

addData (Dataset): With a given String from ".csv" document format, this method reads the line and converts this in a Data object which adds to LinkedList.



compare (Node): With a given double value, this method return true if this value is bigger than node value, otherwise, false. This allows to split data and runs through the tree.



4.2 DESIGN CRITERIA

We found in LinkedList a great data structure for getting and storing a big amount of data with a lower big-O complexity. The static size and numeric access to elements were others reasons why we chose it instead of Stack or Hash Table structures.

We chose a similar C4.5 algorithm using information gain concept according to simple and natural understanding of this algorithm.

Information gain has a big amount of code lines so we created a Gain class with static methods that they do necessary operations to get information gain values.

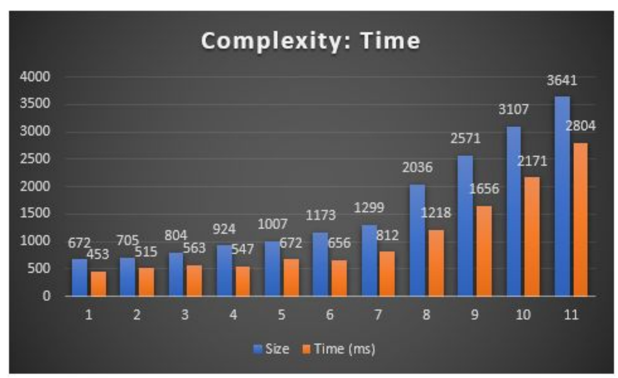
4.3 COMPLEXITY ANALYSIS

Method	Complexity
setLabel	O(log n)
createTree	O(n^3)
addData	O(1)
compare	O(1)

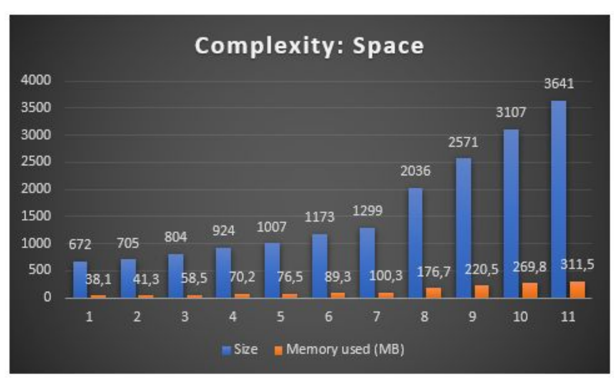
We can see that createTree is the most complex and the slowest method, but other operations are very simple.

In next graphics, size refers to dataset size.

4.4 EXECUTION TIME



4.5 MEMORY USED



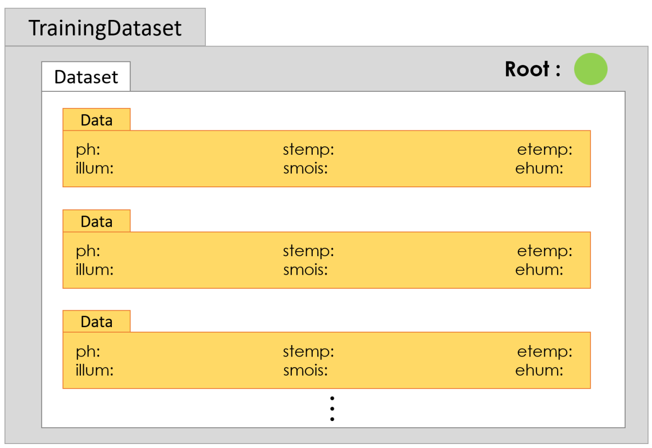
4.6 RESULT ANALYSIS

Previous plots measure addData and createTree methods execution only. We see that the algorithm to create a Dataset can be late a lot if we give it more data, but it is a good time to create even the decision tree.

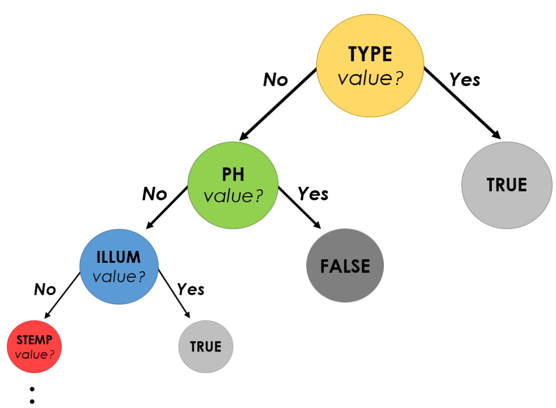
5. "SELECTION TREE"

This is the given name for our data structure. We might say that it does not change significantly in regard to our previous data structure because the changes were focused more in the UML internal structure and some descriptive names.

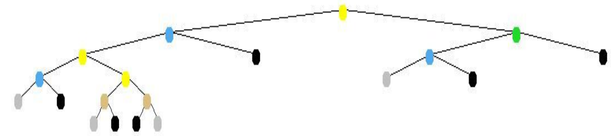
As an example, now we have TrainingDataset and Dataset Java classes. The first is related to the decision tree set and training data. The second one is a data LinkedList that, by definition, is an element from the TrainingDataset class:



We concluded that this binary tree has the less possible number of nodes. Maybe a ternary tree will have a better space complexity, but a bigger time complexity. We prefer this implementation.



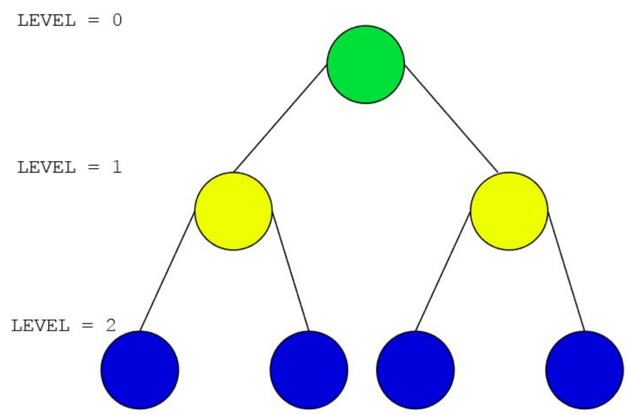
In this point we add a new class, which prints the tree using Java Graphics. Our repository has the implementation.



5.1 OPERATIONS

setLabel (Data): No significative changes from previous data structure.

createTree (TrainingDataset): This belongs to TrainingDataset now and it sets a level for each node.



This creates the tree like previous data structure.

addData (Dataset): A dataset is a data LinkedList now, so addData belongs to this class. Even so, this still is like previous definition.

compare (Node): This method works equal than previous data structure.

5.2 DESIGN CRITERIA

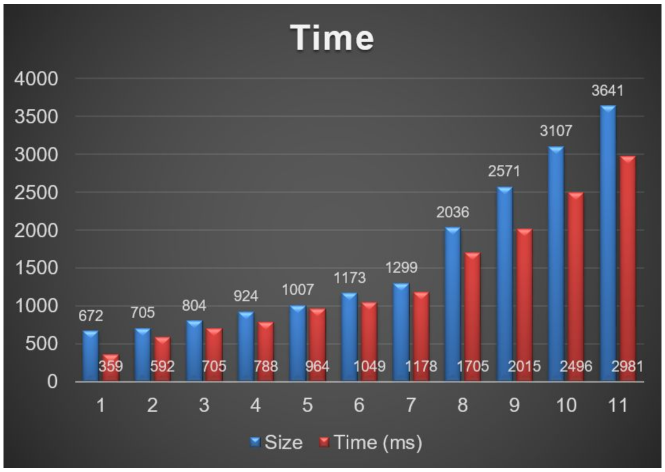
We changed some elements of our code to improve its comprehension. We improved the abstraction and facilitated new methods implementation.

5.3 COMPLEXITY ANALYSIS

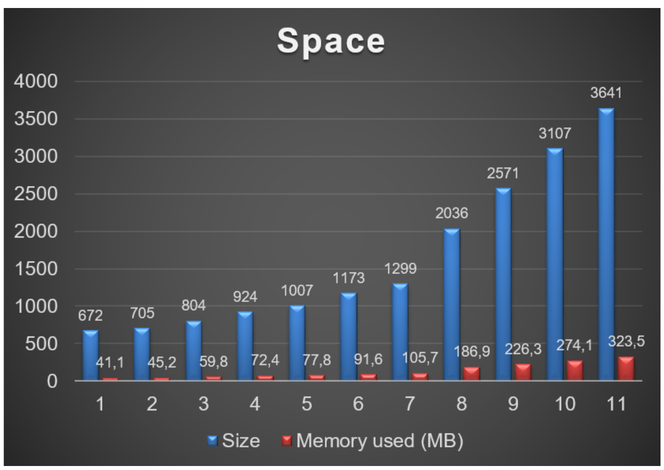
Method	Complexity
setLabel	O(log n)
createTree	$O(n^3)$
addData	O(1)
compare	O(1)

We can see that createTree is the most complex and the slowest method, like previous data structure, but other operations are very simple.

5.4 EXECUTION TIME



4.5 MEMORY USED



4.6 RESULT ANALYSIS

The results obtained by calculating the time and memory consumption of the final executable file developing with the same magnitude as in the previous data can conclude the amount of memory needed and the time it takes to execute is not very unequal we show in previous results.

This may happen because the changes made after the first results are not of great magnitude compared to those already implemented.

6 CONCLUSIONS

This data structure selects a label for any plant from dataset with high accuracy and effectivity, for saving time of big amounts of people.

However, technological advantages really simplify the problems. Day by day, we see how computational engineering through data structures and algorithms works shoulder by shoulder to improve life quality.

Even so, this document does not pretend to be the exclusive solution. At opposite, it pretends to be a started point to continue improving this algorithm. We hope that we have known better implementations soon.

6.1 FUTURE WORK

Maybe ternary trees could be a good implementation, but they affect code speed. Of course, its implementation will give new ideas to implement a good solution, but that topic will be in another document.

Artificial intelligence allows to automatizate a lot of processes. Neural networks and other solutions works very good in this topic. To change the decision tree implementation for others AI implementations will be a very good practice.

7 ACKNOWLEDGEMENTS

This research supported by Coursera's course parte del curso de Minería de datos dictado por el profesor Karim Pichara Baksai en Coursera [6]. Su explicación de la ganancia de información y la entropía es muy sencilla y digerible. Fue con esos dos conceptos con los que pudimos crear nuestro árbol con unas excelentes particiones.

REFERENCES

[1] F. Caparrini, *Inductive Learning: Decision Trees*. (December, 2018). Available: https://bit.ly/2TmPNza. [Accessed: August 9, 2019].

[2] Croplife, *Coffee leaf rust*. Available: https://bit.ly/2w9OCbL. [Accessed: August 9, 2019]

[3] R. Eckhardt, *Introduction to the CART Model*. (9 Feb 2017). Available: https://bit.ly/2KrmfNO. [Accessed: August 9, 2019]

[4] B. López, *C4.5 Algorithm*. (December 2005). Available: https://bit.ly/2pcFG17. [Accessed: August 9, 2019]

[5] M. Mehta, A. Rakesh, and J. Rissanen, *SLIQ: A Fast scalable classifier for data mining*. Available: https://bit.ly/31Ak801. [Accessed: August 9, 2019]

[6] K. Pichara, 2019. *Introducing to data mining*. Available: https://bit.ly/2MTLZUl. [Accessed: October 28, 2019]

[7] scottjulian, *MyC45*. [Java file]. (6 March 2015). Available: https://bit.ly/2TobRte. [Accessed: August 10, 2019]

[8] J. Vincent, 2019. *The state of AI in 2019*. (28 Jan 2019). Available: https://bit.ly/2Te2RFY. [Accessed: August 9, 2019]