COFFEE RUST DETECTION FOR CATURRA VARIETY USING DECISION TREES

Paulina Pérez Garcés
Universidad EAFIT
Colombia
pperezg@eafit.edu.co

Santiago Cartagena Agudelo
Universidad EAFIT
Colombia
scartagena@eafit.edu.co

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

ABSTRACT

Coffee is one of the main plant-based goods consumed today, as well as the second most traded commodity in modern economy after oil. Its production takes place mostly in developing countries such as Colombia, where a malfunction during the process could affect thousands of coffee farmers. Coffee rust does just that, reduce the quality and quantity of grains devastating entire productions yearly. [1]

One of the most effective ways to prevent this disease from spreading is an early detection, which makes effective the extraction and replacement of withered crops. Using wireless sensor networks, it is possible to obtain data from each plant's conditions; which allows us to determinate whether a specific plant is in the early stages of infection or perfectly healthy.

Serving this purpose, a data mining and machine learning concept called decision trees was implemented. With the help of one of these alone we were able to correctly classify 79% of the plants that were studied. In search for a better result a new method was used; one called random forest where multiple trees are built and the most common result is taken as the final one.

Keywords

Coffee Rust; Decision Trees; Data Structures; Big O Notation; Complexity; Caturra Coffee; Machine Learning.

ACM CLASSIFICATION KEYWORDS

Applied computing → Computers in other domains → Agriculture

Applied computing → Life and medical sciences → Bioinformatics

1. INTRODUCTION

Coffee has been one of the most popular drinks since the 1800s, when all of eastern British colonies began growing it in large quantities. Around 1879 coffee farmer in Sri Lanka (formerly called Ceylon) noticed that a new fungus called Hemileia vastatrix was affecting their plantations, but an efficient cure was never found.

The losses caused by this disease made it unsustainable to keep growing coffee in those regions; therefore, plantations of a single healthy strain -Arabica coffee- were moved to the Americas, which quickly became the biggest productors of this good. The crops there were constantly checked, and remained rust-free until 1970, when spores from the fungus were found in Bahia, Brazil. [2]

Since arabica coffee was the most planted strain in America, and due heavy winds and rainy seasons, rust spread around the continent in a matter of years. Countries where the plant had become one of the main tradable products began suffering substantial economic losses.

2. PROBLEM

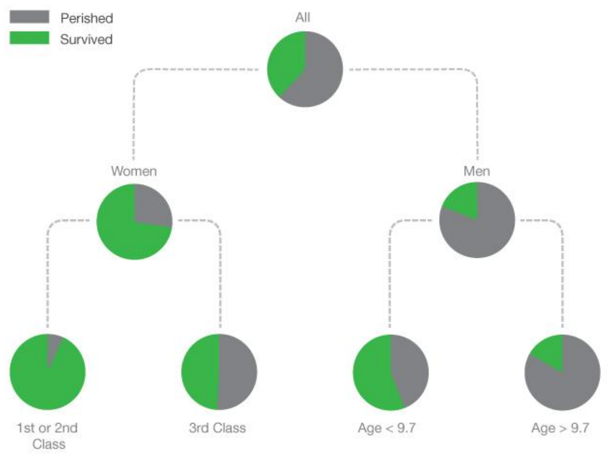
The intent of this project is to create an algorithm capable of determinate whether a coffee bush is vulnerable and suffering from rust or not. These results will be based on data such as soil temperature, weather temperature, soil's PH, illuminance and others; all taken from wireless sensor networks placed in a greenhouse in University EAFIT in Medellín, Colombia.

Said algorithm will be making decisions using methods known as decision trees. This approach uses statistics from the previously classified crops to identify new affected plants, in more technical words, it is a supervised learning method used for classification and regression. [3]

3. RELATED WORK

3.1 CART

CART is an alternative decision tree algorithm. It can handle both classification and regression tasks. This algorithm uses a metric called "Gini Index"; whose objective is to create decision points for classification tasks. CART is similar to C4.5 but there is a notable difference between them: CART constructs the binary trees based on numerical splitting, whereas C4.5 include the step of rule sets. The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any, should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions. [4]



[5]

3.2 CHAID

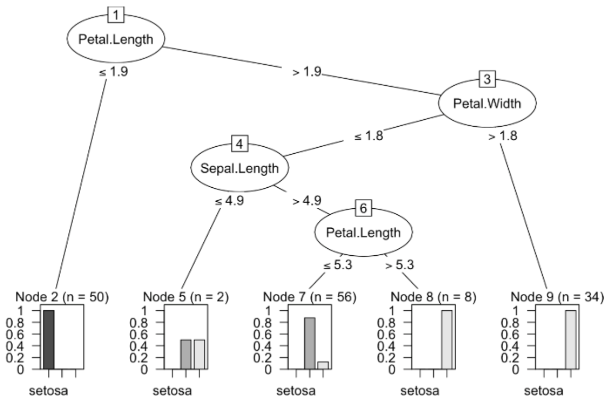
This algorithm, short for Chi-squared Automatic Interaction Detector is one of the oldest (1980) and most popular decision trees. It is a simple set of rules that lets you add more than two tree branches on a single node. The name, as well as some of its principles, comes from the Chi-Squared Statistic that expresses the relationship between two or more non-numerical variables. This technique can easily analyze large amounts of data, but that would also create big amounts of divisions inside the tree. [6]

3.3 ID3

One of the most popular algorithms is the ID3 algorithm, also known as the Iterative Dichotomiser 3. It is used to generate a Decision Tree from a dataset. ID3 is also considered as a precursor to C4.5, as well as many other decision tree algorithms. A big advantage of using ID3 algorithm is that it builds the fastest but also the shortest trees; this makes more understandable the problem. Just as all the algorithms used for building decision trees it has a couple of disadvantages, the principal one being that if you want to implement ID3 algorithm is expensive to train it, meaning you need a lot of data to do so. [7]

3.4 C5.0

C5.0 algorithm is the successor of C4.5, developed by Ross Quinlan in 1993. It splits decisions from the one that gives more information to the ones that can easily be removed or ignored. When the test data comes through, it predicts a single category to which it belongs.; these classes into which the data fall can be discrete attributes divided into a set of intervals. C5.0 trees can be very useful when data is missing from datasets, because it works relatively well without it. The main differences with its processor are the speed, memory usage and size of the trees. [3][8]



[9]

3.RED-BLACK TREE

In computer science a Red-Black Tree is an abstract type of data, specifically is a BST(Binary search tree), there are many kinds of BST but most of them are used for predicting selections or samples of whatever, this is not that case, the Red-Black tree is a different kind of BST, this one has the objective to store data, the best part is that this kind of BST also search, insert and delete data in O(log(n)) complexity, where the letter “n” is the quantity of elements in the tree.

If you want to work with a Red-Black tree you must consider this is a kind of tree in which every node has a color attribute, there are two options: Black or Red, also you need to remember that leaves of any Red-Black tree do not contain relevant data in general, besides you must follow some properties or some rules when you are implementing this kind of BST, those important rules are the following ones:

- As we mentioned before, every node is going to be black or red, but not both or another color.
- All the null leaves must be black.
- Every red node must have two more black tree children.
- Is mandatory the tree root to be black.

4. RED-BLACK TREE

Binary search trees are really efficient when you care about organization; their creation and modifications are not complicated and follow a simple logic: right if the data is bigger, left otherwise. Additionally, new data will cause a “reorganization” for the properties of this tree to stay the same.

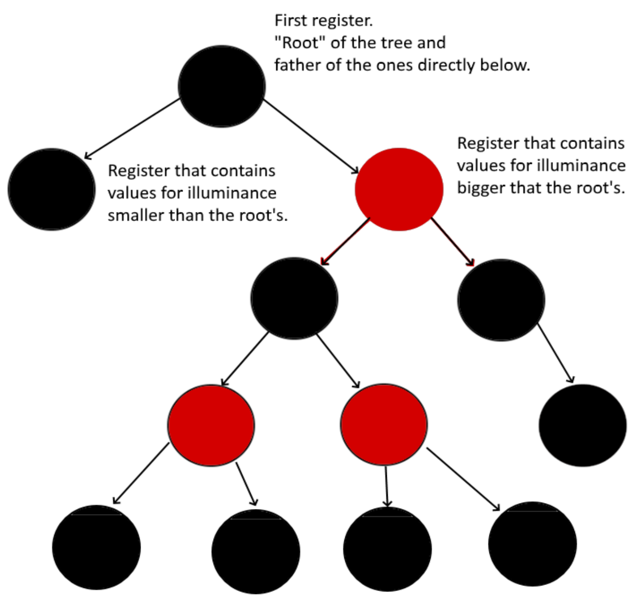

First register.
"Root" of the tree and
father of the ones directly below.
Register that contains values for illuminance smaller than the root's.
Register that contains values for illuminance bigger that the root's.

Figure 1: Red-Black tree structure.

4.1 OPERATIONS OF THE DATA STRUCTURE

4.1.1 Insertion of a register to the tree:

When inserting new data to the tree the structure of it may vary a little bit so it stays balanced; but it doesn't change what it does in principle, search for the place of a new value based on the data that is smaller or bigger.

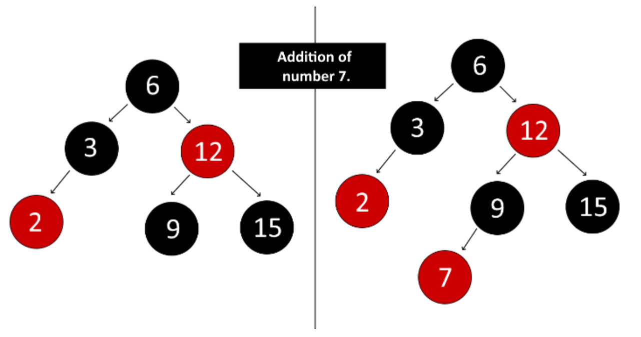


Figure 2: Addition operation pictured with simple numeric data.

4.1.2. Removal of a register from the tree:

This might be trickier than adding a new element but mainly it is the same thing. Once it deletes the desired value, the data will reposition itself to

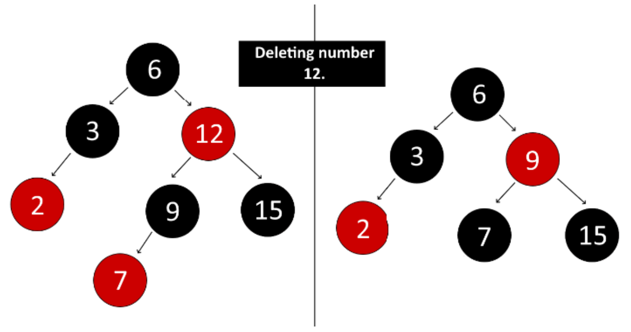


Figure 3: Deletion operation pictured with simple numeric data.

4.1.3. Searching for a value inside the structure:

Once the structure is known, checking if a value is inside the tree is fairly simple. Following the truth/false statements about the size of a value one is able to find a value.

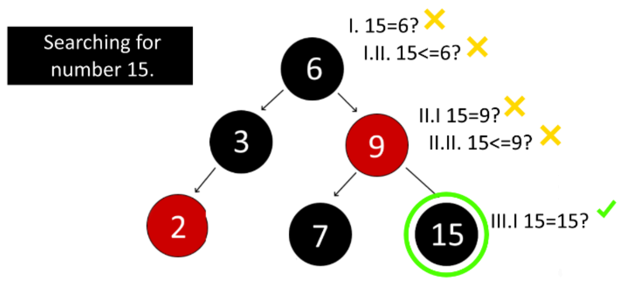


Figure 5: Search in a Red-Black Tree

4.2 DESIGN CRITERIA OF THE DATA STRUCTURE

Red-Black Trees are efficient when thinking about time and space. Two things that are really valuable due their shortage and importance. It is also a really understandable data structure that, when show graphically, can make anyone understand the basis of how it works.

4.3 COMPLEXITY ANALYSIS

METHOD	***COMPLEXITY***
Search	O(log(n))
Insertion	O(log(n))
Removal	O(log(n))

Table 1: Table to report complexity analysis

4.4 EXECUTION TIME

OPERATION	TIME FOR TRAINING DATASET	TIME FOR TESTING DATASET
Creation	23.5 ms	17.3 ms
Insertion	9.8 ms	10.6 ms
Removal	4.5 ms	6.4 ms
Search	3.2 ms	5.4 ms

Table 2: Execution time of the operations of the data structure for each data set.

4.5 MEMORY USED

	MEMORY USAGE FOR TRAINING DATASET	MEMORY USAGE FOR TESTING DATASET
Creation	84.2MB	82.7MB

Table 3: Memory used for each dataset.

4.6 RESULT ANALYSIS

The data structure used for the project works as fast as man can hope for, with a complexity lower than others. The organization is easy to understand, as well as useful. Memory usage could be improved, but the results weren't bad. A Red-Black tree continues to be one of the most efficient data structures.

EFFICIENCY	FOR 100 REGISTERS	FOR 150 REGISTERS
TIME	23.5 s	11.2 s
MEMORY	84.2 MB	84.4 MB

Table 4: Result Analysis for used data structure

COMPLEXITY	STACK	LINKED LIST	HASH TABLE	RED-BLACK TREE
SEARCH	O(n)	O(n)	O(n)	O(log(n))
INSERT	O(1)	O(1)	O(n)	O(log(n))
DELETE	O(1)	O(1)	O(n)	O(log(n))

Table 5: Complexity of different data structures.

5. RANDOM FOREST APPLIED THROUGH DECISION TREES THAT IMPLEMENT THE CART ALGORITHM.

In contrast to red-black trees, binary decision trees are not for storing data; these are used to study and predict the behavior of a dataset. This is done by using nodes as questions sorted by the information gain we can obtain from them. Now, a random forest is a set of trees which evaluate the dataset in different ways to obtain various solutions and, using the majority, gain certainty.

5.1 OPERATIONS OF THE DATA STRUCTURE

5.1.1 Gini index calculation:

This type of operation is called a splitting measure, it helps us find how relevant a split will be when building a tree. This one in particular indicates the chances of classifying data wrong when randomly chosen.

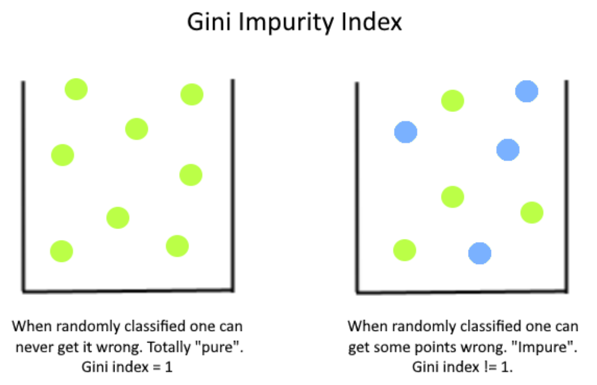

Gini Impurity Index
When randomly classified one can never get it wrong. Totally "pure". Gini index = 1
When randomly classified one can get some points wrong. "Impure". Gini index != 1.

Figure 6: Graphic explanation of Gini Index

5.1.2 Information gain calculation:

In this case it would be very easy to infer or recognize the function that this operation fulfills, it tells us how much uncertainty is there left in the algorithm if a particular attribute and value are chosen for the split. Therefore, the attribute with the maximum Information Gain, also known as an entropy measure, is chosen for searching the best split.

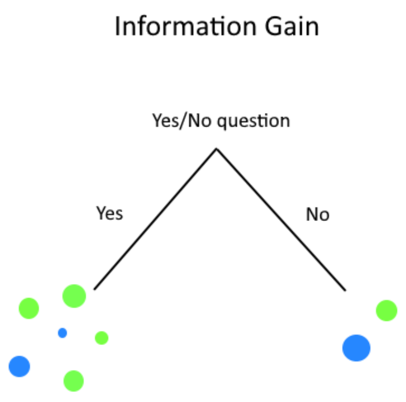


Figure 7: Representation of the information gain.

5.1.3 Finding the best data split:

The find_best_split operation was mentioned on the previous one operation, because this one takes the attribute with the highest information gain and then proceeds to search for the value that must divide the data. Intuitively, we can say that the best split is the question where we can separate the classes more productively based on that best feature.

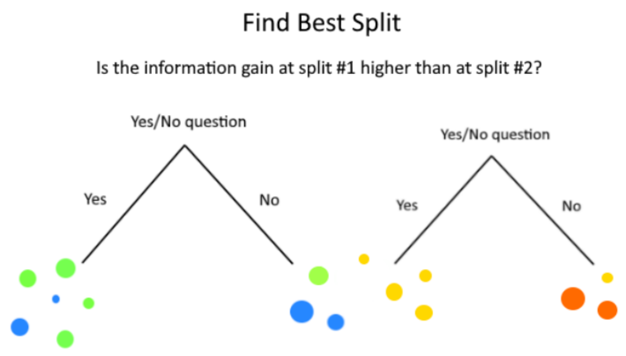

Find Best Split
Is the information gain at split #1 higher than at split #2?
Yes/No question
Yes
No
Yes/No question
Yes
No

Figure 8: Best Data Split

5.1.4 Partitioning the data:

The partition operation does an important work, because this one makes a classification between those values that can answer affirmatively to a question (truth values) and those that cannot (false values). This is the reason why this method can be called binary decision tree, the answer to a question must always be true or false.

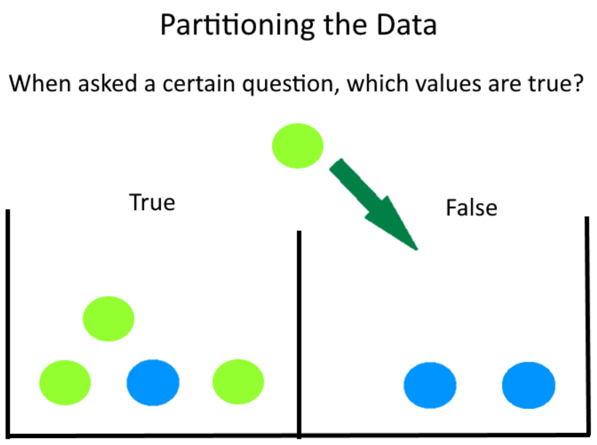


Figure 9: Partitions

5.1.5 Building the tree:

Finally, we have this operation which uses the operations previously mentioned so it can use their answers for laying the foundations of the tree. This process is done recursively until the information gained is none, where we know that the leaves of the decision tree have been reached.

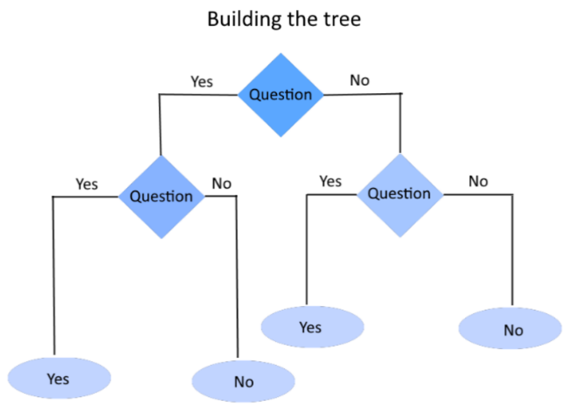


Figure 10: Building the tree

5.2 Design criteria of the data structure: Decision Trees are considered "one of the best and mostly used supervised learning methods." (Brid, R. S; 2018) they are easy to understand, since the tree analogy is fairly simple, as well as fast and mostly accurate. Another advantage of decision trees that implement the CART algorithm is that they can do classifications as well as regression for the given data. One of its biggest flaws is overfitting, but with the help of methods such as random forest it stops being an overwhelming issue. The simplicity of it makes it easier for us to work and adapt them to our circumstances. After seeing that a single tree had an accuracy of almost 80% it became clear that it was a viable and efficient method.

5.3 Complexity analysis

Taking into account the following notation, take a look of the chart about the complexity analysis in the algorithm:

- N represents the labels.
- M represents the rows.
- K represents the columns.
- L represents the values.

METHOD	COMPLEXITY
GINI INDEX	O(n*m) + O(n) = O(n*m)
INFORMATION GAIN	O (1) + O(n*m) = O (n*m)
BEST DATA SPLIT	O(k*l) + O (1) +O(m*n) + O(m) = O(k*l) + O(m*l) + O(m)
PARTITIONING	O(m) + O (1) = O(m)
BUILDING THE TREE	O(m) + O(k*l) + O (1) +O(m*n) + O(m) = O(k*l) + O(m*l) + O(m)

Table 6: Table to report complexity analysis

5.4 Execution time

METHOD	TRAINING DATA	TESTING DATA
GINI INDEX	0.2s	0.4s
INFORMATION GAIN	0.5s	0.7s
FIND BEST SPLIT	0.6s	0.8s
PARTITIONING	0.1s	0.1s
BUILDING THE TREE	1.5s	1.5s

Table 7: Execution time of the operations of the data structure for each data set.

5.5 Memory used

TRAINING DATA	TESTING DATA
120 MB	160MB

Table 8: Memory used for each operation of the data structure and for each data set data sets.

5.6 Result analysis

The fast times obtained for each method, as well as the simple implementations make decision trees a perfectly suitable method for working this problem. Its complexities do not consume more time than they should and, even though it can be improved, the ratio accuracy-complexity is just right for a problem such as ours.

6. CONCLUSIONS

Coffee Rust is a plant disease that has deeply affected plantations all around our country and our continent. Its effects on economy and harvests have made people hurry in search for a solution but so far only early prevention has been significantly effective. Due development of new technologies and studies, machine learning algorithms such as decision trees have been declared as successful tools, useful for this kind of problems. Keeping these facts in mind we were set to write some code for solving said problem.

The results were incredibly good. Even when at first, we were a little lost and worked with data structures designed to save data instead of classifying new one, everything worked out just fine. At the end, when we got a hold of decision trees and how to build them, we obtained over 75% accuracy with a single tree alone. In search for better results random forest were implemented, where accuracy rates must have improved. It all made sense and solved our coffee rust problem once we started implementing the new and efficient methods.

Nevertheless, some things can still be improved. More advanced techniques that could be implemented are still out there waiting for being discovered or implemented. Some problems such as the possible overfitting of the algorithms must be solved and worked on. Even so, this project gave us inspiration and courage to keep on learning and work towards better results.

6.1 Future work

For future research and as a way of improving the obtained results similar projects can be made using more and more complex data as well as implementing more specialized data structures and actions such as branch pruning.

ACKNOWLEDGEMENTS
The writers of this article would like to thank fellow EAFIT students Alejandro Salazar and Andrés Grimaldos as well as the University EAFIT for their support while working on this project.

REFERENCES

1. Goldschein, E. 11 Incredible Facts About the Global Coffee Industry. *Business Insider*. 2011. https://www.businessinsider.com/facts-about-the-coffee-industry-2011-11?op=1#according-to-the-world-wildlife-fund-of-the-50-countries-with-the-highest-deforestation-rates-from-1990-to-1995-37-were-coffee-producers-11

2. Arneson, P.A. Coffee rust. *The Plant Health Instructor*. 2000. https://www.apsnet.org/edcenter/disandpath/fungalbasidio/pdlessons/Pages/CoffeeRust.aspx

3. Scikit-learn Documentation. Decision Trees. https://scikit-learn.org/stable/modules/tree.html

4. Brownlee, J. Classification And Regression Trees for Machine Learning. *Machine Learning Mastery*. https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/

5. The Fact Machine. http://www.thefactmachine.com/wp-content/uploads/2014/05/CART-TreeMap6Node.jpg

6. Statsoft. Popular Decision Tree: CHAID Analysis, Automatic Interaction Detection. http://www.statsoft.com/Textbook/CHAID-Analysis

7. Howe, B. Building Trees: ID3 Algorithm. Coursera. https://es.coursera.org/lecture/predictive-analytics/building-trees-id3-algorithm-7PNNn

8. IBM. C5.0 Node. https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.modeler.help/c50node_general.htm

9. Kuhn, M; Ross, Q. Plot a decision tree. https://topepo.github.io/C5.0/reference/plot.C5.0.html

10. Brid, R. J. *Introduction to Decision Trees.* (2018) Available at: https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb