Wireless sensors for detecting rust in caturra coffee: Data structures for the prediction of infected crops.

Juan Pablo Ossa Zapata
Eafit University
Colombia
jpossaz@eafit.edu.co

Mauricio Jaramillo Uparela
Eafit University
Colombia
mjaramillu@eafit.edu.co

Mauricio Toro
Eafit University
Colombia
mtorobe@eafit.edu.co

ABSTRACT
The objective of this project is to analyze and propose a possible solution to the late detection of Roya, one of the most catastrophic plant diseases in history, present in coffee crops in several Latin American countries, including Colombia. In order to do so, an algorithm implementation is proposed, that through the study of data collected by a network of wireless sensors is able to analyze and predict which crops have or are likely to have this fungus. The solution to this problem is of paramount importance to the Colombian economy because more than half a million families depend on these crops for their livelihoods. For this reason, it is our responsibility to contribute to the development of technologies and methods capable of reducing the impact of this infection in the countryside of our country.

Categories and Subject Descriptors
F.2 [**Design and analysis of algorithms**]: Data structures design and analysis

General Terms
Coffee Rust, Roya

Keywords
Pattern matching, Data structures design and analysis

1. INTRODUCTION
Colombia, a country recognized for its great variety of crops and the quality of its products abroad, has been constantly threatened by the presence of a fungus that has been affecting one of its most internationally desired products: coffee. Composed of more than 563,000 families approximately, the guild of coffee growers makes possible the export of 13.5 million bags of coffee a year, thus achieving coffee to be the main agricultural export product of the country. However, this product has been going through very critical times due to a pest known as Roya which, due to its late diagnosis, is very difficult to treat. In search of a solution to this problem, a network of wireless sensors was implemented to maintain a set of coffee crops with constant monitoring, where physical and chemical data related to the appearance of this fungus was be collected. The task at hand is therefore to implement efficient algorithms and data structures that allow classification and understanding of the collected data.

2. PROBLEM
The problem we face is based on creating, through the use of data structures, a system capable of relating the data already studied of Caturra coffee plants achieving to establish parameters and possible causes that make the Rust appear in coffee crops, so as to know beforehand if there is the presence of this fungus in the studied crop. To achieve this purpose will represent a great advance for the Colombian agriculture, through its implementation in the cultures of Caturra coffee to diminish the high quantity of cultures lost by cause of the Rust.

3. RELATED WORK
We present the following related work as the base material for the construction of our own solution to the problem.

3.1 ID3 algorithm
ID3 is an algorithm to generate a decision tree created by Ross Quinlan focused on the search for hypotheses or rules based on a set of examples formed by a series of continuous data called attributes in which one will be the attribute to classify. This, also known as objective, is of binary type, that is, it will have values such as positive or negative, yes or no, valid or invalid, etc. The ID3, based on the previously entered examples, tries to obtain the hypotheses by means of which to classify new instances in positive or negative.

3.2 C4.5 algorithm
This algorithm, developed by Ross Quinlan, is an extension of the ID3 algorithm mentioned above. C4.5 constructs decision trees from a set of training data in the same way that ID3 does, using the concept of information entropy. At each tree node, the algorithm chooses a data attribute that divides the set of samples into subsets as efficiently as possible. In this way, the attribute with the highest gain of normalized information is chosen as the decision parameter.

3.3 CART algorithm
CART is a technique with which classification and regression trees can be obtained. When the target variable is

discrete, classification is used; when it is continuous, regression is used. This algorithm finds the independent variable that best separates our data into groups, expressing it as a rule to assign its corresponding node. Then, for each of the resulting groups, the same process is repeated recursively until it is not possible to obtain a better separation.

3.4 CHAID algorithm

CHAID is a classification method for generating decision trees by chi-square statistics to identify optimal divisions. It was proposed by Gordon V. Kass in 1980 and is currently one of the most used in marketing studies.

4. DESIGN OF THE DATA STRUCTURE

More than just trying to provide a solution to the main problem itself, we allow one of our goals to be that the implementation of the final algorithm is optimized to allow high volumes of data input. This implies designing a data structure, that the algorithm can use efficiently, such that it takes into consideration the nature of the operations performed.

4.1 Building blocks of the data structure

We present the following *'building blocks'* as the set of basic abstract units that allow the representation of the proposed data structure, as well as an explanation of each one of them:

- `DataValue`: A union type that can hold both discrete (as `int`) and continuous (as `double`) values.
- `DatasetFeature`: [figure 1] A structure that holds the relevant information required to describe a feature in the dataset, along with the name of that feature.

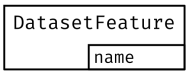

Figure 1: Visual representation of the `DatasetFeature` element.

- `DatasetHeader`: [figure 2] A structure that describes the features of the dataset, as well as the label that the algorithm will seek to optimize.

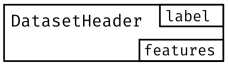

Figure 2: Visual representation of the `DatasetHeader` element.

- `DatasetEntry`: [figure 3] A node in the `Dataset` linked list. It contains an ordered, fixed-size, set of `DataValue` elements. The size of the set is determined by the number of features in the `DatasetHeader` of the associated `Dataset`.
- `Dataset`: [figure 5] This is the main substructure. It is essentially a doubly linked list that holds references to the both the head and tail of a chain composed of `DatasetValue` objects. This structure also has one reference to a `DatasetHeader`. Some relevant properties arise from the design of this block:

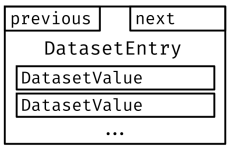

Figure 3: Visual representation of the `DatasetEntry` element.

  - Fast $[O(1)]$ insertion with a reference. This allows loading data from disk with the maximum possible performance.
  - Full iteration capability. No random access is required, which means that the linked list nature of the structure does not affect the overall performance.
  - Definition of sub-datasets without increasing the memory usage. As seen in figure 4, two `Dataset` objects can have `head` and `tail` elements in the same chain, thus allowing the definition of sub-datasets without memory redundancy.

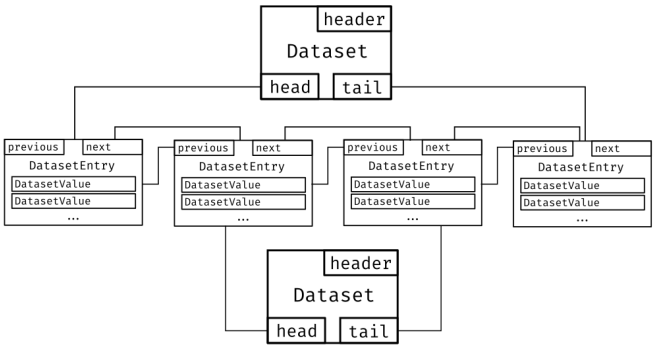

Figure 4: Visual representation of the *sub-dataset* concept.

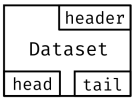

Figure 5: Visual representation of the `Dataset` element.

4.2 Example data structure usage

Consider a dataset with the features shown in table 1 and the data shown in table 2. Using the proposed data structure, this example dataset would have a similar appearance to that described in figure 6.

4.3 Complexity analysis of the data structure

An analysis of data structure's time complexity provides further information on it's fitness for solving the task. Table 3 presents the worst case time complexity for common tasks that the algorithm performs, as well as an explanation on the parameters for said complexity.

name	type	label
fruit	discrete	yes
red	continuous	no
green	continuous	no
blue	continuous	no

Table 1: Features of the example dataset.

fruit	red	green	blue
1 (apple)	246.45	10.32	1.23
1 (apple)	235.23	30.75	3.94
2 (orange)	217.24	184.23	11.23
2 (orange)	250.01	120.88	0.42

Table 2: Data of the example dataset.

4.4 Data structure benchmarking

We provide some additional statistics on the exection time and the memory usage of the `C` implementation of the data structure:

- Table 4 and figure 7 show the time used while loading the dataset from disk.
- Table 5 and figure 8 show the time used while loading the dataset from disk.
- Table 6 and figure 9 show the time used while determining a split position that provides the best information gain.
- Table 7 and figure 10 show the memory used while determining a split position that provides the best information gain.

The fact that the memory usage is the same for all datasets can be explained by the way memory is handled by the program. Unless the difference in sizes of datasets is considerably high, the memory usage will remain the same for all of them. Also, the reason why the dataset `rust` takes a longer period of time to find a split position is explained by the fact that the algorithm determined that a higher precision is required for considering a split. This happens because the difference between the upper and the lower boundary of one or more of the features is higher than in the other two datasets.

5. REFERENCES

[1] CropLife Latin America. Roya del Cafeto. `https://www.croplifela.org/es/plagas/listado-de-plagas/roya-del-cafeto`

[2] Charris, L Henríquez, C, Hernández, S, Jimeno, L, Guillen, O, Moreno S. *Análisis comparativo de algoritmos de árboles de decisión en el procesamiento de datos biológicos.*

[3] Centro de Estudios y Aplicaciones Logísticas. Faculty of Engineering of the National University of Cuyo. *Algoritmo ID3.*

[4] Bosco, J. *Árboles de decisión con R clasificación.*

[5] IBM. *Nodo Chaid.*

[6] Quinlan, J. R. *C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.*

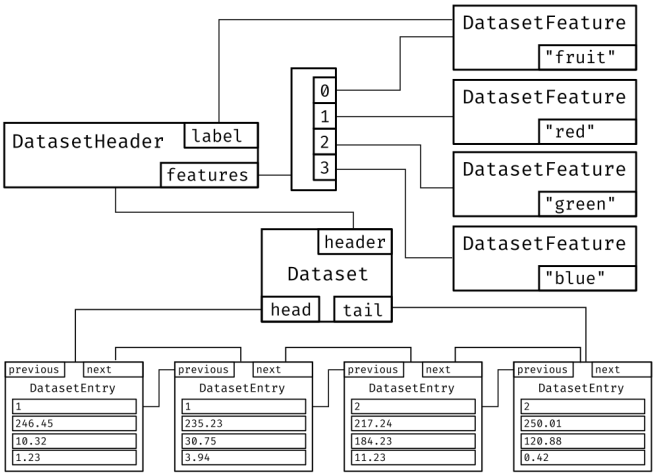

Figure 6: Example dataset appearance.

Task	complexity	parameters
Loading from disk	$O(n)$	• n: The number of entries to be loaded from disk.
Find best information gain split	$O(mno)$	• m: The number of entries in the dataset. • n: The number of features in the dataset. • o: Variable proportional to the precision used for split evaluation.

Table 3: Table of complexity for common tasks.

Dataset	time (seconds)
rust	0.00229582
rust-test	0.00109157
rust-train	0.00108636

Table 4: Time used while loading the dataset from disk.

Dataset	memory (Kb)
rust	964
rust-test	964
rust-train	964

Table 5: Memory used while loading the dataset from disk.

Dataset	time (sec)
rust	3.3113
rust-test	0.2223
rust-train	0.2233

Table 6: Time used while finding split position.

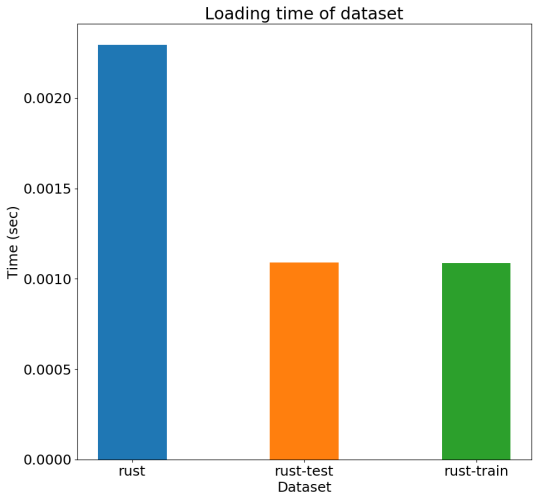

Figure 7: Time used while loading the dataset from disk.

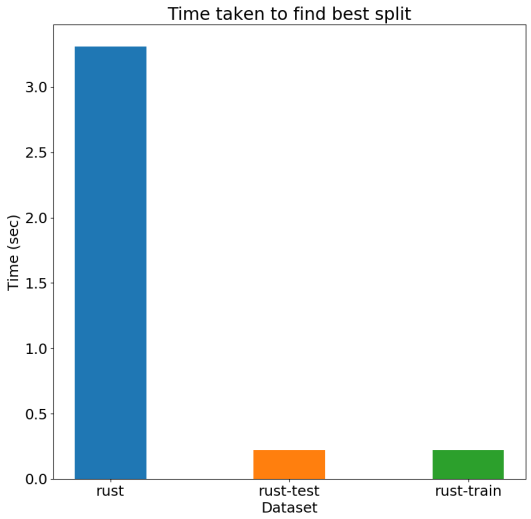

Figure 9: Time used while finding split position.

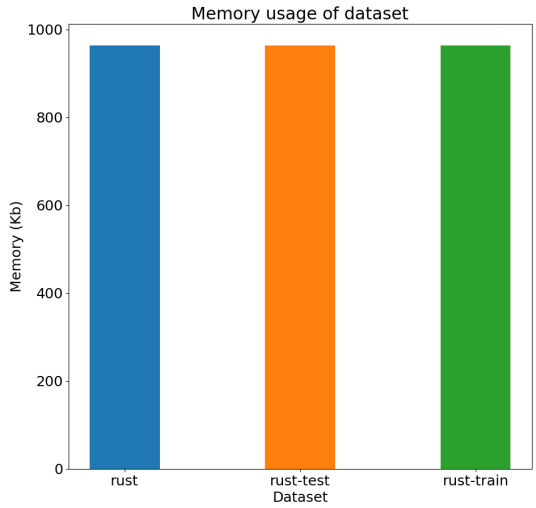

Figure 8: Memory used while loading the dataset from disk.

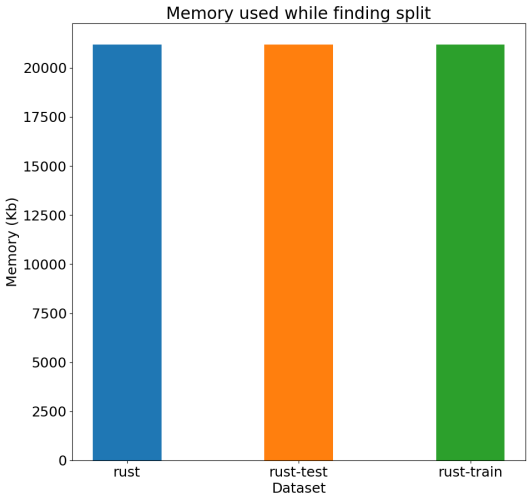

Figure 10: Memory used while finding split position.

Dataset	memory (Kb)
rust	21196.3
rust-test	21196.3
rust-train	21196.3

Table 7: Memory used while finding split position.