DOCUMENTATION

Random Forest Accelerator

Mohamad Baha Eddin Kabaweh(222252)
Kassem Abbas(222516)
Sanad Marji (223090)

Course: Digital Design for Machine Learning
Supervisor: Mikail Yayla

Contents

1. Introduction

Machine Learning is a technique of data analysis and automation of analytical model building. It's a branch of Artificial Intelligence based on the concept that computers can recognize patterns, learn from data, and make decisions with little to no human input.

Decision Trees are supervised Machine learning algorithms most frequently used for both Regression and classification problems, and is heavily applied in various industries such as E-Commerce and banking to predict outcomes and behavior. On the other hand, a Random Forest is a supervised machine learning algorithm made of a collection of decision trees bundled together as one Entity.

The implementation of a random forest accelerator hardware was designed specifically for the IRIS-Dataset, and is implemented in „VHDL". For the sake of the implementation,
Scikit-Learn was used to implement the random forest model. In the course of this documentation, the subject of decision trees and random forest will be examined in more detail.

2. Random-Forest model using Scikit-Learn

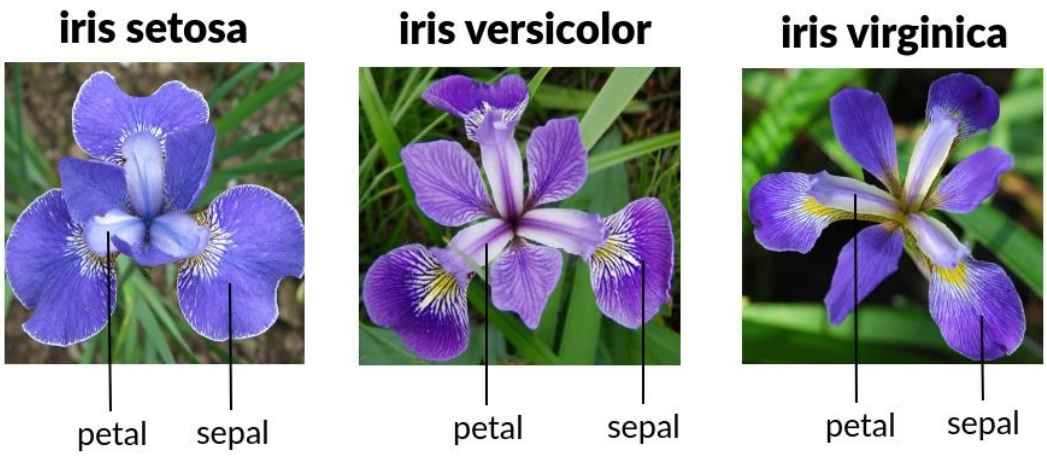
Using Scikit-Learn, the Random Forest model was implemented in python in order to be used in the Hardware Design.

Scikit-Learn is a Python module that provides tools for machine learning and data mining tasks. It includes implementations of many algorithms used in pattern recognition, regression analysis, dimensionality reduction, clustering, and outlier detection.
The Random Forest model provides all the information needed to build the different decision trees that form the Forest and also provides information about the nodes of each tree (Class, Threshold, Feature, Samples, Gini).

More information on the Random Forest Classifier using Scikit-Learn can be found here.

2.1 IRIS-Dataset

The Iris flower data set is a multivariate data set. It includes 50 samples from each of three Iris species (Iris virginica, Iris setosa, and Iris versicolor). Each sample has four characteristics measured: the length and width of the sepals and petals in centimeters. The purpose behind this project is to implement a random forest accelerator fitted for the IRIS-Data set.



Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3	1.4	0.1	Iris-setosa
14	4.3	3	1.1	0.1	Iris-setosa
15	5.8	4	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa

Figure 1.1: IRIS-Data Set

2.2 Quantization

Since the IRIS-Dataset contains the thresholds and features of the training and test data as floating point values, the data requires some quantization in order to store the values at lower bit-widths than the original floating-point-bit-widths. It also allowed for a more straightforward, compact, and higher performance model to be used for the computations coming with the risk of losing some of the accuracy when calculating the final outputs/classifications.

Quantization maps a floating point value x∈[α,β] to a b-bit integer xq∈[αq,βq]

$$\mathrm{xq} = \mathrm{round}(\frac{1}{c}\mathrm{x} - \mathrm{d})$$

where: $\mathrm{c} := \frac{\beta-\alpha}{\beta\mathrm{q}-\alpha\mathrm{q}}$ and $\mathrm{d} := \frac{\alpha\beta\mathrm{q}-\beta\alpha\mathrm{q}}{\beta-\alpha}$ $and\ z := round(\frac{(\beta*\alpha q)-(\alpha*\beta q)}{\beta-\alpha})$

More concretely, the quantization process will have an additional clip step.

$$\mathrm{xq} = \mathrm{clip}(\mathrm{round}(\frac{1}{c}\mathrm{x} + \mathrm{z}), \alpha\mathrm{q}, \beta\mathrm{q})$$

where clip(x, l, u) function is defined as

$$clip(x, l, u) = \begin{cases} \mathrm{l}, \mathrm{if\ x} < \mathrm{l} \\ \mathrm{x}, \mathrm{if\ l} \leq \mathrm{x} \leq \mathrm{u} \\ \mathrm{u}, \mathrm{if\ x} > \mathrm{u} \end{cases}$$

The features and thresholds of the IRIS-Dataset (range of values between 0.1 … 7.9) can be represented using 4 bits. The Quantization mapping of a floating point value x to y-bit integer value is checked using the trial and error method, where the goal is an achieved accuracy of more than 80%.

The implemented Quantization-function is also executed in the program and the resulted quantized values substituted the default values (features and thresholds) of the Random Forest model.

2.3 Architecture of Random Forest Accelerator

To load the quantized data-stream from Python to Random Forest Accelerator, the values should be stored/written inside of a .txt file.

The stored data represents each node of each Decision Tree in the Random Forest.
Each node is an Std_logic_vector(19 downto 0) 20 bit -vector and has the following form:

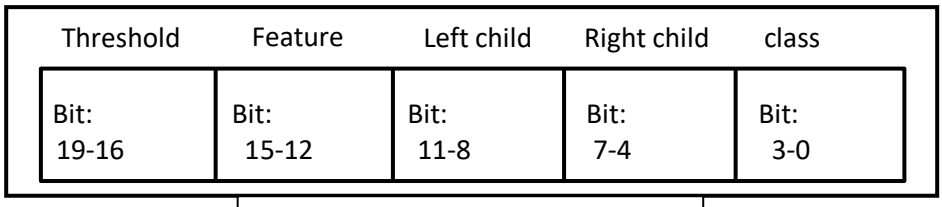


Figure 1.2: Basic Node Structure

Each node in the tree is represented as a 20-bit Data Stream, where Each 4 bits in such a data stream represent an attribute in that node.

A Decision Tree is made up of Nodes (more specifically range of 1-15 nodes in each decision tree) and has the following form:

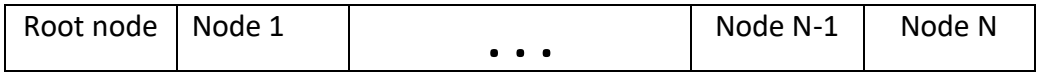


Figure 1.3: Basic Decision Tree Structure

A Random Forest is a collection of different Decision Trees (more specifically 10 DT's) and has the following form:

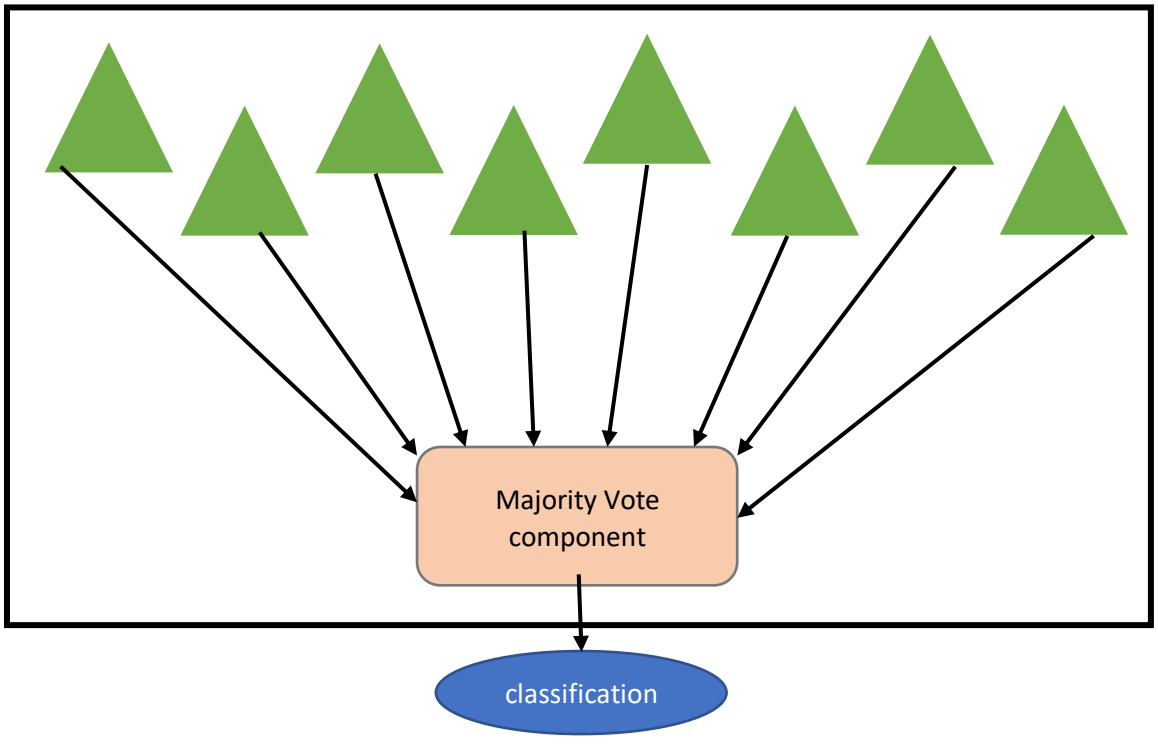


Figure 1.4: Random Forest Structure

3. Random Forest Accelerator

The Random Forest Accelerator is the heart and main design of the hardware implementation. It creates instances of different decision trees, stores each classification/decision in an array. The Majority vote component then gives out the most frequent class.

To understand the structure of the whole design, here is a summary of the different components used:

- Random forest Accelerator
    - Decision Tree
        - Node
            - Comparator
        - DT_memory
    - Majority vote

Each of the above-mentioned hardware designs will be discussed separately and in more detail during the course of the documentation.

3.1 Decision Tree

The Decision Tree design in the Random Forest Accelerator generalizes the structure of all decision trees found within the random forest, in other words it can build any tree within the structure. The component has 5 inputs:

1. Tree_number: specifying the tree that should be traversed
2-5. Sepal_length, Sepal_width, Petal_length, Petal_width: Features/input values of one iris flower that are needed for the classification.

And one output

1. Class_out: classification of the decision tree according to the input values.

Depending on the max_depth of the decision tree the number of "node" components used within the design is = max_depth. The implemented design uses a total of 3 "node" as components, since the max_depth of all decision trees within the random forest classifier is 3 (max_depth 3 achieves highest accuracy in classification using the IRIS Data set, which was determined using trial and error method while testing, a depth of more than 3 was found to be redundant and achieved the same accuracy and in some cases less). ***

Through the use of "node" component the "Decision Tree" can iterate and traverse without any complications. At the end the Class_out is fetched from the leaf node reached and a classification is given.

Another component used within the DT design is the "DT_memory" where the data stream containing the attributes and all info of each node in the decision tree is fetched.

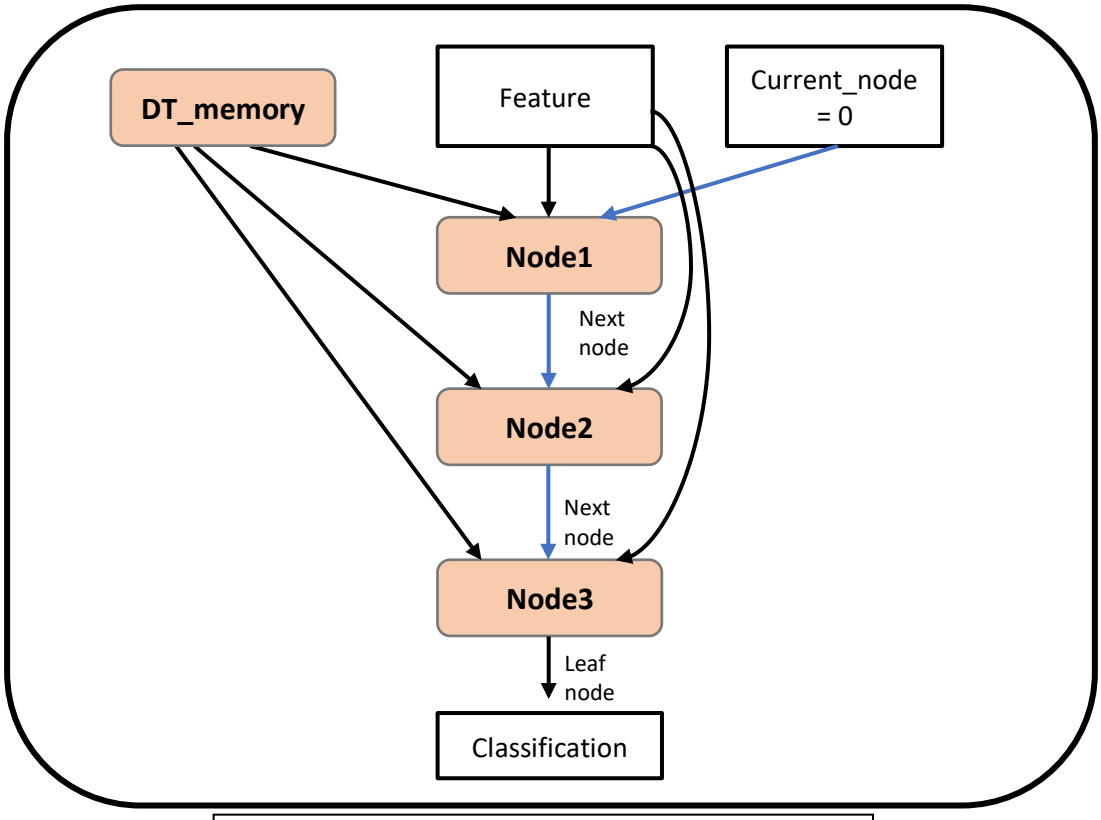


Figure 1.5: Hardware Design Decision Tree Structure

*** an increase in depth of Decision Tree leads to more complicated design and implementation***

3.2 Node

The Node component is a hardware component representing a single node in a decision tree. It has 3 inputs (all_info, feature_to_compare, current_node) and one output(next_node). the purpose of the "node" is to do some calculations and comparisons using its inputs and characteristics (feature and threshold) in order to indicate what the **next node** will be.

Basically it facilitates the traversal through the nodes of the tree until the feature value (all_info[15 downto 0]) of the node is "1111" which describes a leaf node and thus returns the address of itself as the "next_node". Traversing through the tree is the main functionality in a decision tree, and doing so correctly will return a correct classification.

The "node" uses a "comparator" as a component within its design, it feeds the comparator a feature and its threshold and expects a certain result.
When the result is:

- Greater_than = '1'
    - Next_node stores the address of the **right child** of the current node
- Less_than_equal = '1'
    - Next_node stores the address of the **left child** of the current node

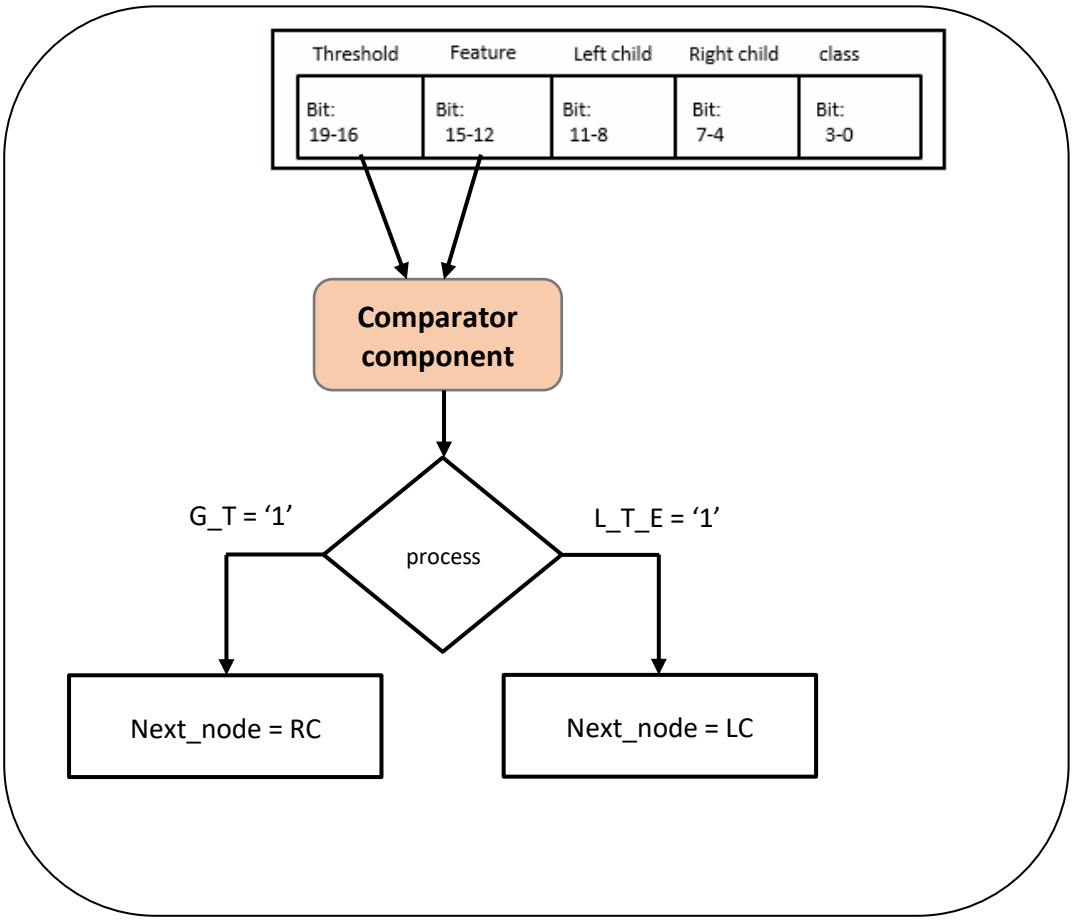


Figure 1.6: Hardware design Node Structure

3.3 Comparator

The comparator has 2 Inputs (feature and threshold) and 2 Outputs (greater_than, less_than_equal). It compares the feature with the threshold.
If feature is greater than the threshold, then the comparator assigns the “greater_than” = 1 and “less_than_equal” = 0. Otherwise the outputs will be inverted.

Figure 1.7 shows the structure of the comparator:

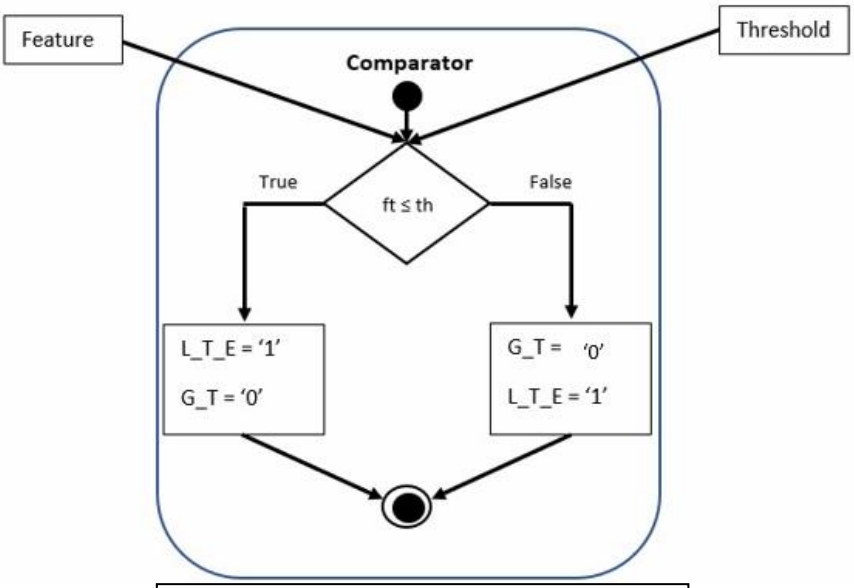


Figure 1.7: Comparator Structure

3.4 DT_memory

A 2-dimensional array used to store all information needed for all the decision trees in the random forest. The Function InitRamFromFile is used to read data stream from a .txt file generated by the python code(in Github) and store the data stream in the array. There are a total of 10 data chunks within the .txt file, each one of the 10 data chunks can hold 1-15 20-bit data streams each representing information of a node.

```
1  10110000000101100010      Node 0 tree 0
2  01100001001001010000      Node 1 tree 0
3  10100000001101000001      .
4  00001111000000000000      .
5  00001111000000000001      .
6  00001111000000000000      .
7  10010010011110000010      .
8  00001111000000000001      .
9  10100010100110100010      .
10 00001111000000000010      .
11 00001111000000000010      Node 10 tree 0
12 x
13 01010010000100100000      Node 0 tree 1
14 00001111000000000000      .
15 00110011001101000010      .
16 00001111000000000001      .
17 10010010010101100010      .
18 00001111000000000010      .
19 00001111000000000010      .
```

snippet of the .txt file and the following table corresponds to how its stored / represented in the memory.

	Nodes in Tree													
Tree0	N0	N1	N2	.	.	.	.	.	.	.	.	.	.	N14
Tree1	N0	N1	N2	.	.	.	.	.	.	.	.	.	N13	
Tree2	N0	N1	N2	.	.	.	.	.	.	.	N11			
Tree3	.	.	.	.	.	.	.	.	.	.	N11			
Tree4	.	.	.	.	.	.	.	.	.	N10				
Tree5	.	.	.	.	.	.	.	.	N9					
Tree6	.	.	,	.	.	.	.	.	.	.	.	.	N13	
Tree7	.	.	.	.	.	.	.	.	.	.	.	.	.	N14
Tree8	.	.	.	.	.	.	N7							
Tree9	N0	.	.	.	.	N5								

3.5 Majority Vote

The majority vote takes the classifications of all the decision trees within the random forest as an input, specifies what class is the most frequent, and gives it as an output. The majority vote is used within the random forest design.

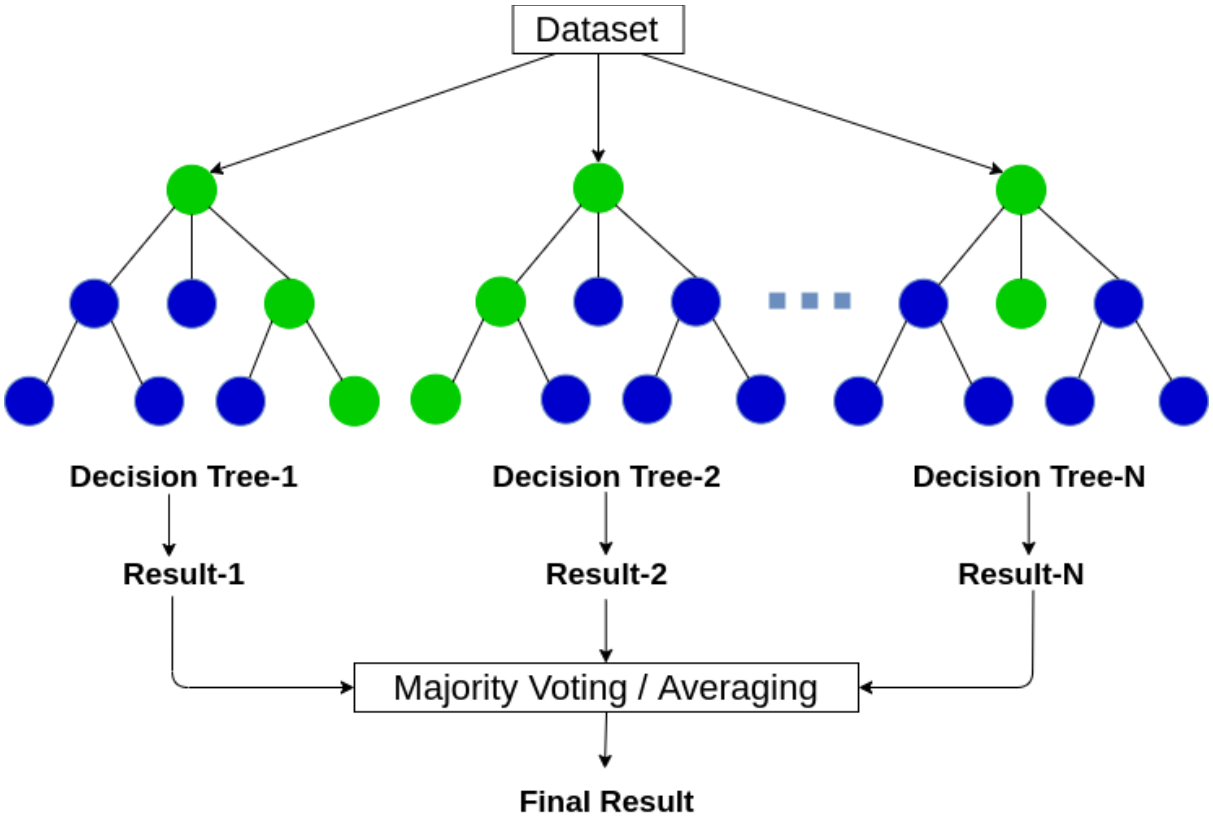


Figure 1.8: Majority vote

4. Conclusion

In conclusion, Random Forest is a great algorithm/model for both classification and regression problems, to produce a predictive model. its default hyperparameters return high accuracy result and the system itself was implemented to avoid overfitting.

In our opinion, doing the “Fachprojekt” remotely was not so bad for us. We had some difficulties at the start of the project but we managed to surpass these obstacles and implement the Random Forest Accelerator on Hardware. The Project was really interesting and kept us intrigued to find out more and all in all was fun to implement.

A random forest accelerator was implemented during the course of the project. The implementation was made in VHDL. The modules were implemented there separately from each other and put together at the end. In the future it would be conceivable to scale the Random Forest Accelerator with larger data sets and more complex structures or to run the Random Forest Accelerator on an FPGA. In addition, there would be the option of having the nodes generated automatically, based on the depth of the decision tree. A further development for the project would be to develop hardware that suits any data set automatically and without any extreme changes to the design (example MNIST-data set). The design can also be expanded through the use of Systolic Arrays which would make it more efficient.