Cellular Automata based Decision Tree.

In this approach the Cells of the cellular automata are modelled with the states of the input data properties.

Example:

Consider the following

ATTRIBUTE	POSSIBLE VALUES		
Weather	Sunny	Cloudy	Rainy
Temperature	Hot	mild	cold
Humidity	High	low	-
Wind	Week	strong	-
Decision	No	Yes	-

So the Cells in cellular automata will be corresponding to

Sunny	Cloudy	Rainy	Hot	Mild	Cold	High	Low	Week	Strong

This array/matrix is known as the characteristic matrix, and would be referred to as T.

The decision column is used to decide the thresh-hold values.
So in this case the thresh-hold would be 0.5 since there are only two decision classes.
There fore if the net weight for a particular input is less than 0.5 it will fall into the first category and if it is more than 0.5, to the second category.

The input for decision is also represented in the same format as T, but the value of the cell is either a 1, if that property holds, or 0 if that property is absent.

So a typical input for statement : “Sunny weather, mild temperature , low humidity, and strong wind” would be represented as

Sunny	Cloudy	Rainy	Hot	Mild	Cold	High	Low	Week	Strong
1	0	0	0	1	0	0	1	0	1

So “1 0 0 0 1 0 0 1 0 1” forms the cellular automata C_i whose next state needs to be calculated using the characteristic matrix T.

Characteristic Matrix T holds continues values, which are adjusted during the training phase to accommodate for the error.

This approach involves two phases:

1. Learning phase.
2. Classification phase.

Learning Phase:

Algorithm:

1. Initialize all cells in the Characteristic matrix to initial value in [0,1].
2. For all training instance i, create $C_{i.}$
3. For all i do.
4. Multiply the row matrix C_i with the column Matrix T. (Since multiplication is a 1xN Nx1, where N is the number of class properties or number of cells in the T matrix, only one value is obtained Y_i).

5. Compare Y_i with the threshold and determine class.
6. If given input does not fall in the required class, then adjust weights.
7. For adjusting weights a simple mean error is calculated as
   meanError = (class – Y_i)/totPropertyClasses.
   Where *class* is the value for the expected decision class and *totPropertyClasses* is the number of properties. In the above case it would be 4 as there are 4 properties which are being checked for, namely Weather, temperature, humidity and windy.
8. Add the *meanError* to weights which are active for that particular input i.
9. Repeat for all test cases.

With the sample data (input.dat) the Characteristic matrix before and after the learning phase are shown below.

Sunny	Cloudy	Rainy	Hot	Mild	Cold	High	Low	Week	Strong
0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
0.059375	0.606250	0.000000	0.175000	0.137500	0.353125	0.012500	0.353125	0.234375	0.131250

Classification Phase:
In classification phase again the data is represented as C_i and process of matrix multiplication with the charecteristic matrix T is carried out to obtain Y_i, which then is compared with the thresh-holds values to determine the class to which the given input belongs to.

Eg:

C_i :

Sunny	Cloudy	Rainy	Hot	Mild	Cold	High	Low	Week	Strong
1	0	0	0	1	0	0	1	0	1

After multiplying with the characteristic matrix we get Y_i as 0.481250 which is less than the threshold of 0.5, therefore gets classified to the first class, which in this case is ***No***.

Thresh hold values are calculated based on the number of Decision classes, in case of three classes in Decision class, the thresh hold values would be 0.66666 and 1.3333332 respectively for classifying the input cases.

Improvements Over existing system:

The existing system works by deciding classification states based on entropy and information gain, which is calculated with respect to the amount of available inputs. The proposed system follows the neural network approach of adjusting weights to tend the system towards the required output.

Other features of this approach are:

1. Parallelism: Since the input rows are modelled on cellular automata, the decision algorithm can be distributed among multiple systems and the decision classes can be computed in parallel.
2. Independence: The decision made by different properties are independent from each other and can be as mentioned earlier, can be spawned across multiple systems compute the decision faster.
3. Complexity: The complexity of the given system is linear in nature, which is approximately

equal to totPropertyClasses * i, where i is the number of training instances.

4. Incomplete Data: Decision can be made in case a certain property class is missing. The obtained decisions are almost accurate.
5. Massive Pre-Computation: As the decision making is practically a matrix multiplication, a lot of decision inputs can be pre-calculated just by creating a matrix of inputs, which can be used for faster decision making, at a later stage.

Areas to be improved:

The current system does the error correction using a meanError variable, this process of weight adjustment can be improved to arrive at better converged weights which would help in generating more accurate results.
Initial weights need to be assigned more optimally, instead of constant weights so that the learning phase may be able to generate better weights for each input.

Explanation of the attached code and input:

The code takes input from a file. The input file is divided into three sections namely

1. CL (class)
2. TR (Training)
3. TE (Testing/Classification)

Lines beginning with ! are ignored.
Lines beginning with ~ mark the beginning of a particular section.
'D' represents the decision class.

Apart from file-reading, input string formatting and array creation, the important functions related with the given technique are

1. makeClass : Make class initializes the characteristic matrix, and also sets the decision classes and the thresh-holds associated with each decision class.
2. getClass : getClass does the matrix multiplication to retrieve the class. Since it is a row into column multiplication direct multiplication of the cells is possible.
3. adjustWeight : adjustWeight is the core of this implementation, this function is concerned with weight correction.