# CS5131 Programming Assignment 2

Jed Lim, Karimi Zayan, Mayukh Das, and Matthew Kan

NUS High Schol of Maths and Science

## 1 Background

Breast cancer is the number one most common cancer amongst women in Singapore. Early and accurate diagnosis of breast cancer is important for breast-saving and life-saving treatment.

The gold standard for the diagnosis of breast cancer is by surgically removing the breast lump with a complete microscopic examination of the breast tissue to look for cancer cells.

Fine needle aspiration is an alternative that allows the doctor to take out a small amount of tissue from the breast lump, without the need for surgery to remove the entire breast lump. By examining the characteristics of the cells, doctors have been able to diagnose breast cancer with variable success. Increasing the success of fine needle aspiration allows for diagnosis of breast cancer without the need for a woman to undergo surgery to remove the breast lump.

To resolve this, this project uses a fuzzy decision tree to classify breast tumor cells into malignant cancer cells or benign non-cancerous cells.

## 2 Dataset

The dataset used was the Breast Cancer Wisconsin (Diagnostic) Data Set from the University of Irvine (UCI) Machine Learning Repository. The dataset contained 569 instances, with no missing data. 357 instances were benign (not cancerous) and 212 were malignant (cancerous).

The features were computed from digitalized images of fine needle aspirates of breast tumors. They describe 10 characteristics of the cell nuclei present in the images:

- The radius of an individual nucleus
- The nuclear perimeter
- The nuclear area
- Compactness of the nucleus
- The smoothness of the contour of the nucleus
- The number of contour concavities
- The symmetry of the nuclear contour
- The texture of the cell nucleus

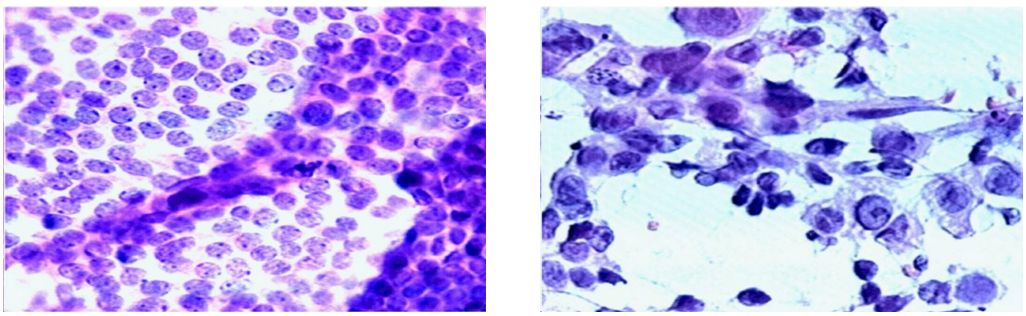


**Fig. 1.** A picture of breast cells. The cells on the left are begin while the cells on the right are cancerous.

The mean, standard error and worst (mean of the three largest values) of these features are computed for each image, resulting in 30 features in the UCI dataset.

We follow in the footsteps of Sizilio et al. **??** and add newly generated features of homogeneity and uniformity that were demonstrated to have diagnostic importance.

Uniformity is the difference between the radius worst value and the radius mean value and is an indication of the variability in size of the cell nuclei.

Homogeneity is the difference between the worst value of symmetry and the mean value of symmetry and is an indication of the symmetry of the cell nuclei.

According to Sizilio et al., the features of area, perimeter, homogeneity and uniformity produced the best results. Thus, we use these 4 feature and drop all other features.

## 3 Decision Tree

To solve this problem, we will make use of a decision tree.

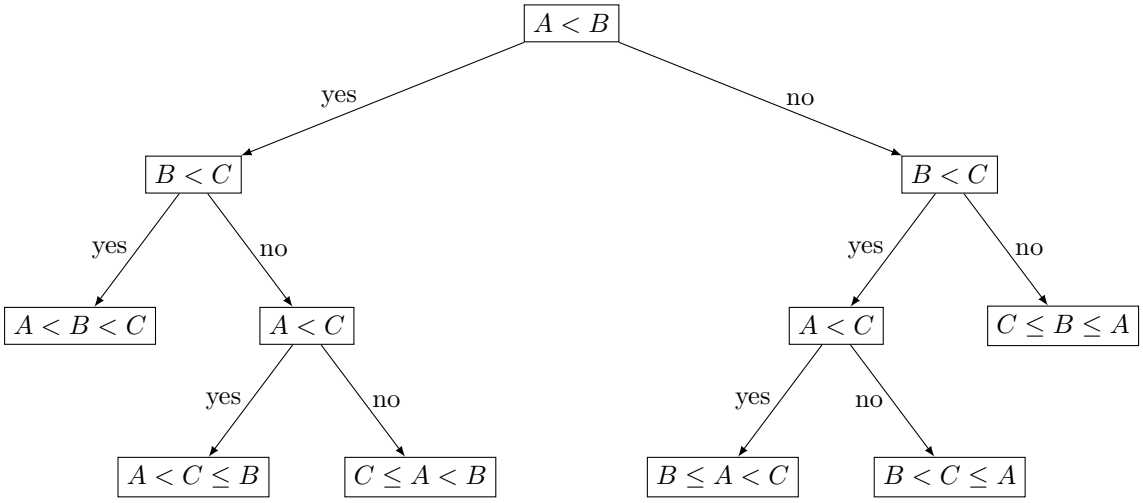



**Fig. 2.** A decision tree for sorting three values.

Entropy A decision tree essentially asks certain questions at the right times, in order to determine which class the input belongs to. To do this, a value known as the entropy with the following formula is used.

$$E = \sum_{i=1}^{C} p_i \log_2 p_i$$

It measures the "impurity" of a set of objects. When the set is homogeneous (i.e. all the objects are of the same class), the entropy will be low. When the set is heterogeneous (i.e. the objects are from different classes), the entropy will be high.

Information Gain In order to decide what question to ask at which point in time, we need to determine which question results in the greatest decrease in entropy. This is calculated using the information gain shown below:

$$IG = 1 - E$$

The feature that results in the greatest information gain will be selected for that branch of the tree. For continuous features, a threshold will be chosen based on what threshold will result in the greatest information gain.

Finally, we need to explain when the decision tree will decide when it should stop splitting. It will do this when a maximum depth $d$ has been reached or when there a fewer than $s$ samples remaining to split.

Pseudocode With this, we can now construct our decision tree. The pseudocode for construction and inference can be found 3.

- $\mathbf{S}$ is the training data with $m$ classes and $n$ continuous features. It is a set and all $\mathbf{S}$ are within $\{\mathbf{S} : \mathbf{S} = ((f_1, \ldots, f_n), c), f_i \in \mathbb{R}, c \in [1, m]\}$
- $d$ is the maximum depth that the tree can reach and $s$ the minimum size of the data before we stop branching the tree.
- We also define the node object $\mathbf{N}$. It has a children attribute $N.\mathsf{children}$ which is a list of $\eta$ child nodes. $\{\mathbf{N}_1, \ldots, \mathbf{N}_\eta\}$. Furthermore, it also stores a function $\mathbf{N}.\mathsf{func}((f_1, f_2, \ldots, f_n))$ which takes in a list of features and outputs an integer $i \in [1, \eta]$. The depth of the node in the tree is also stored in $\mathbf{N}.\mathsf{depth}$

Time Complexity Now, let us analyse the time and space complexity of our algorithm.

- Let us assume that the tree will be fully filled with height $d$. This means that there will be $2^{d+1} - 1$ nodes in the tree. This means the split would have been done $2^d - 1$ times. In practice, this may not be true but let's assume so for the sake of simplicity.
- Splitting the tree has a time-complexity of $O(n \times |\mathbf{S}|^2)$ where $|\mathbf{S}|$ is the length of the data that still remains at that node and $n$ is the number of features. This is because calculating the entropy has time-complexity $O(|\mathbf{S}|)$ and there $|\mathbf{S}|$ thresholds to compute and $n$ features to compute them for.
- Now,let's denote the size of the dataset at depth $x$ and at the $i^{th}$ node as $|\mathbf{S}_{x,i}|$. We know that $|\mathbf{S}| = \sum_{i=1}^{2^x} |\mathbf{S}_{x,i}|$.
- On average, we can assume that $|\mathbf{S}|$ will be halved at each split. Thus,

$$|\mathbf{S}_{x,i}| = \frac{|\mathbf{S}|}{2^x}$$

- Then, the total average time-complexity for training the decision tree is

$$O(\sum_{x=0}^{d} \sum_{i=1}^{2^x} n \times |\mathbf{S}_{x,i}|^2) = O(\sum_{x=0}^{d} n \times \frac{|\mathbf{S}|^2}{2^x}) = O(n \times |\mathbf{S}|^2 \times (2 - 2^{-d})) = O(n \times |\mathbf{S}|^2)$$

- In the worst case, let's say that the dataset is split unevenly such that all the sample except one go to the next node. Then, $|\mathbf{S}_{x,0}| = |\mathbf{S}| - x$ and $|\mathbf{S}_{x,1}| = 1$.
- Then, the total worst-case time-complexity for training the decision tree is

$$O(\sum_{x=0}^{d} \sum_{i=1}^{2^x} n \times |\mathbf{S}_{x,i}|^2) = O(\sum_{x=0}^{d} n \times (|\mathbf{S}|^2 + 1)) = O(n \times d \times |\mathbf{S}|^2)$$

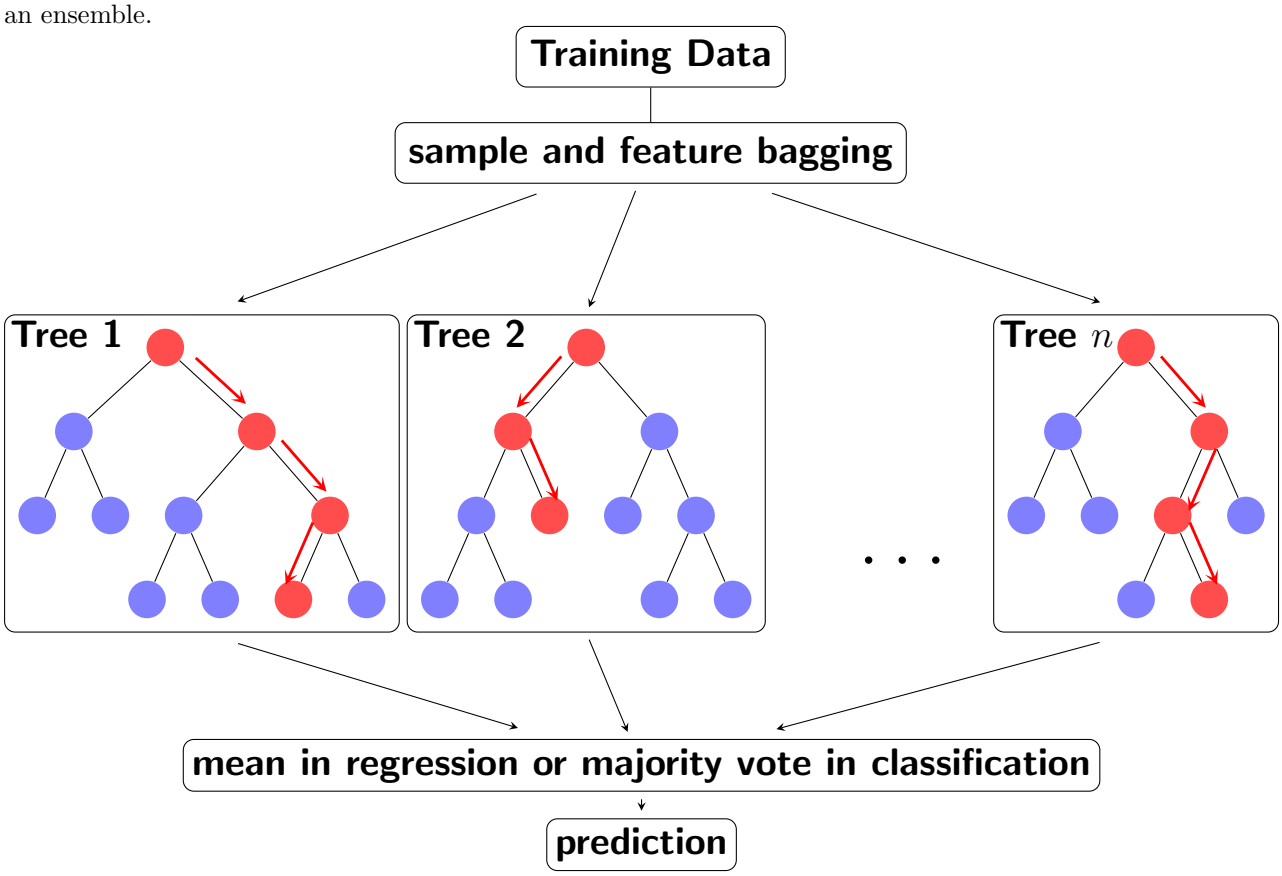
- Now, let's calculate the memory complexity of training the tree. Calculating the entropy requires creating the new dataset splits to be handed to the children nodes. This is done in $O(|\mathbf{S}_{x,i}|)$ space because the size of 2 new dataset splits must add up to the dataset they were split from. For simplicity, let's assume none of this memory is deallocated. Then, the memory complexity is

$$O(\sum_{x=0}^{d} \sum_{i=1}^{2^x} |\mathbf{S}_{x,i}|) = O(d \times |\mathbf{S}|)$$

- Now, let's calculate the time complexity of inference. Deciding which path to take in a decision tree is O(1). Thus, for a tree of height $d$ and average height $\bar{d}$, the worst-case time-complexity is $O(d)$ and $O(\bar{d})$. The memory complexity is $O(1)$ because no additional space is needed during inference other than the space needed to store the tree.

## 4 Random Forest

To improve the performance of our classifier, we make use of many different decision tree classifiers in an ensemble.




In essence, the data is split into multiple non-overlapping parts. A decision tree is trained on each part and together they form an ensemble. To obtain the final prediction, majority voting is used on the outputs of each decision tree. If more decision trees predict that the cells are benign, the prediction of the random forest will be benign. On the other hand, if more decision trees predict that the cells are malignant, the output of the random forest will be that the cells are malignant.

## 5 Performance

Now, we shall evaluate the performance of our model. The results are summarised in the table below.

## 6 GUI

## 7 Reflections

MATTHEW Coding the decision tree from scratch was enjoyable and it allowed me to have an under the hood understanding of an important machine learning algorithm. An improvement that we could have made as a group was to do more planning in the beginning. We had to reverse some decisions to improve on our final product. Despite this, my group worked very well together as a team as we are good friends to begin with (and somehow managed to be in the same team even though it was RNG). Overall, I have always liked data structures and algorithms and this module was both educational and enjoyable.

JED I was largely responsible for writing the report and coding out the GUI. From writing the report, I learnt alot about about writing good and clear pseudocode as well as writing good mathematical analysis on time-complexity and space-complexity. For the GUI, instead of using JavaFX, I made use of Jetpack Compose for Desktop. It was much easier to use and it has replaced JavaFX in many projects that I am

doing. Overall, I enjoyed working on this task but I found the need for the tree to used for a social cause very restricting. The task would be easier and much more fun if we have the flexibility of using a tree for whatever purpose we want such as quad-trees / oct-trees for gravitational simulations. Also having more flexibility in what language we can use for the assignment and the module in general would be nice because data structures and algorithms can be implemented in any language, not just Java.

Zayan This project was interesting and challenging. At first, I was tasked to code out the decision tree with the algorithm ID3 (Iterative Dichromatizer 3) with binning. This was challenging to code as it involved a lot of resizing. However, all my work was scrapped because we decided to pivot and used a different decision tree algorithm which Matthew coded out. It was tough seeing your work thrown out but it was a necessary part to put out the best work. I learnt that sometimes in life you have to make tough decisions and redo stuff for the best result. I then coded out the random forest classifier and the metric algorithms and also helped refactor some of the decision tree code. All in all, it was a fun experience.

Mayukh

**Alg** CalculateEntropy$(\mathbf{S}, \mathbf{t})$

$I \leftarrow 0$
$\mathbf{S}_1 \leftarrow \{((f_1, \ldots, f_n), c) : f_i < t, ((f_1, \ldots, f_n), c) \in \mathbf{S}\}$
$\mathbf{S}_2 \leftarrow \{((f_1, \ldots, f_n), c) : f_i > t, ((f_1, \ldots, f_n), c) \in \mathbf{S}\}$
$\mathbf{C}_1 \leftarrow \{c : ((f_1, \ldots, f_n), c) \in \mathbf{S}_1\}$
$\mathbf{C}_2 \leftarrow \{c : ((f_1, \ldots, f_n), c) \in \mathbf{S}_2\}$
$I \leftarrow 1 - \frac{|\mathbf{C}_1| \times E(\mathbf{C}_1) + |\mathbf{C}_2| \times E(\mathbf{C}_2)}{|\mathbf{S}|}$
Return $\mathbf{S}_1, \mathbf{S}_2, I$

**Alg** BuildTree$(\mathbf{S}, d, s)$

Initialise new node $\mathbf{N}$
Initialise empty queue $\mathbf{L}$
$\mathbf{N}_{root} \leftarrow \mathbf{N}$
$\mathbf{L}$.push$((\mathbf{S}, \mathbf{N}))$
While $\mathbf{L}$.size $\neq 0$ do
  $\mathbf{S}, \mathbf{N} \leftarrow \mathbf{L}$.pop()
  If $\mathbf{N}$.depth $> d$ or $|\mathbf{S}| < s$ then break
  $I_{max} \leftarrow 0$
  For $i \in 1 \ldots n$ do
    $E \leftarrow 0$
    For $t \in \{f_i : ((f_1, \ldots, f_n), c) \in \mathbf{S}\}$ do
      $\mathbf{S}_1, \mathbf{S}_2, I \leftarrow$ CalculateEntropy$(\mathbf{S}, t)$
      If $I_{max} < I$ then
        $I_{max} \leftarrow I; i_{max} \leftarrow i; t_{max} \leftarrow t$
        $\mathbf{S}_3 \leftarrow \mathbf{S}_1; \mathbf{S}_4 \leftarrow \mathbf{S}_2$
    End if
  End for
  Initialise new nodes $\mathbf{M}_l, \mathbf{M}_r$
  $\mathbf{M}_l.\mathbf{p} = \frac{|\{c : c == 1, ((f_1, \ldots, f_n), c) \in \mathbf{S}_1\}|}{|\mathbf{S}_1|}$
  $\mathbf{M}_r.\mathbf{p} = \frac{|\{c : c == 1, ((f_1, \ldots, f_n), c) \in \mathbf{S}_2\}|}{|\mathbf{S}_2|}$
  $\mathbf{N}$.func $\leftarrow ((f_1, \ldots, f_n) \mapsto$ If $f_{i_{max}} < t_{max}$ 0 then 1)
  $\mathbf{N}$.children[0] $\leftarrow \mathbf{M}_l$
  $\mathbf{N}$.children[1] $\leftarrow \mathbf{M}_r$
  $\mathbf{L}$.push$((\mathbf{S}_3, \mathbf{M}_l))$
  $\mathbf{L}$.push$((\mathbf{S}_4, \mathbf{M}_r))$
  End for
End while
Return $\mathbf{N}_{root}$

**Alg** Qry$(\mathsf{f}_1, \ldots, \mathsf{f}_n, \mathbf{N})$

$i \leftarrow \mathbf{N}$.func$(\mathsf{f}_1, \ldots, \mathsf{f}_n)$
If $\mathbf{N}$.children$[i] = \perp$ then
  If $\mathbf{N}.\mathbf{p} < 0.5$ then
    Return 0
  else return 1
Else
  Return Qry$(f_1, \ldots, f_n, \mathbf{N}$.children[i])

**Fig. 3.** Pseudocode for construction and inference of the tree.