Stony Brook University
College *of* Engineering
and Applied Sciences

Department of Electrical and Computer Engineering

ESE 589: Learning Systems for Engineering Applications

Project 3: Decision Tree Induction Algorithm

Submitted to Prof. Alex Doboli

Date: 12/09/2021

Team Members:
Hamid Jalili
Mahan Agha Zahedi

Decision Tree Induction Algorithm

Introduction

Classification is a powerful tool used in data analysis that is used to break what was at first glance a single group or raw dataset into categories that we can make inferences about. It has a myriad of use cases such as, for binary cases: credit card fraud detection, medical diagnosis, filtering spam email, etc. For non-binary cases we can break customers into different tiers of spending groups, or classifying different types of fruits and vegetables.

Almost all classification methods can be broken down into two key parts; The learning/training step and the testing step [1]. In an ideal setup, the training tuples are randomly sampled from the data set and the leftover tuples will be used for the testing set, or the unseen data set. The data split between these two is roughly 70% and 30% respectively although the proportions can be different without any noticeable effect on the results. When the trained classification model is created, it is possible for it to overfit the training set meaning that it may pick up some noise that is in the training set but not present in the testing set. We know we have overfit the data when the accuracy of the training set is higher than that of the testing set. Here, the accuracy of a classifier is the percentage of tuples that are correctly classified

Decision Tree Induction is one type of classifier strategy and its implementation will be the focus of this paper. It can handle multidimensional data, is intuitive, and generally offers good accuracy. A decision tree is a flowchart like tree structure, where each non-leaf node designates a test on an attribute, each branch represents an outcome of the test, and each leaf node holds a class label. Given that a decision tree is already constructed, it can be used as a classifier for a given tuple by starting at the root and traversing it to a leaf which will hold the class label.

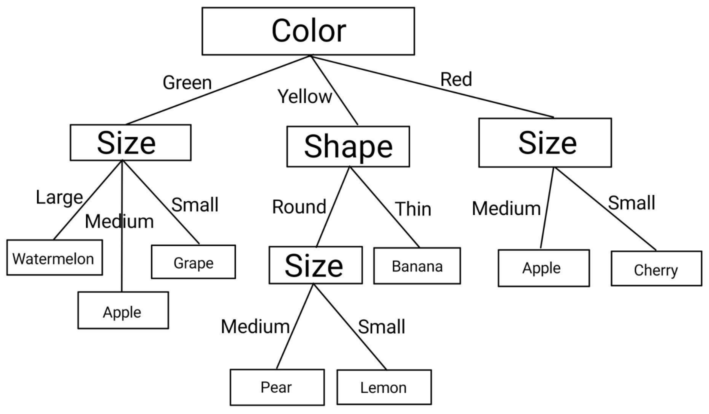


Figure 1: Precomputed Decision tree

Trees are constructed in a top-down recursive divide-and-conquer manner beginning at the root node N. At this node, start with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built. The input parameters of the algorithm are the partitioned data, the list of attributes describing the tuples, and the procedure for selecting the attribute that best describes the given tuple according to class. We want the partitions to be as pure as possible where the purity is defined by how many tuples belong to the same class.

The attribute selection measure provides a ranking for each attribute with the given tuples. The attribute with the best score given the measure is chosen as the splitting parameter. The selection methods are as follows: Gini Index, Information Gain, and Gain Ratio. For Information Gain, the attribute with the highest information gain is chosen as the splitting attribute. The expected info, or entropy of D, is given as

$$Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i) \tag{1}$$

Where p_i is the probability that a tuple in D belongs to class C_i and m is the number of attributes. With

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j) \tag{2}$$

We can measure how much more information is still needed after partitioning to arrive at an exact classification. The smaller this value (the expected information) still required, the greater the purity. The difference of these two values tells us how much information would be gained by branching on attribute A and is aptly called Information Gain.

$$Gain(A) = Info(D) - Info_A(D) \tag{3}$$

The attribute A which maximizes Gain(A) is chosen as the splitting parameter of node N. For continuous attributes, a split point must be chosen, and these values are determined by taking the midpoint between consecutive values.

While the Information gain measure is an effective attribute selection method, it is not without a flaw in that it is biased towards tests with many outcomes. To remedy this bias, the Gain Ratio by applying a normalization to Information Gain by using a “split information” value defined as

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log_2\left(\frac{|D_j|}{|D|}\right) \tag{4}$$

With this the Gain Ratio is defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(A)} \qquad (5)$$

Once again the attribute that maximizes this value is chosen as the splitting attribute. The final attribute selection method we implement is the Gini Index. It measures the impurity of D and is calculated by

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2 \qquad (6)$$

And it considers only binary splits. We compute a weighted sum of the impurity of each resulting partition. If a binary split on A partitions D into D_1 and D_2 , the Gini Index of D is given as

$$Gini_A(D) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2) \qquad (7)$$

For each given attribute, each possible binary split is considered for discrete values. For continuous values we again use midpoint splits. The reduction in impurity that would be incurred by a binary split of attribute A is given as

$$\Delta Gini(A) = Gini(A) - Gini_A(D) \qquad (8)$$

Here, the attribute that minimizes this value is chosen as the splitting attribute. Again, if the attribute is a continuous value, the midpoint of consecutive values are used as a splitting point

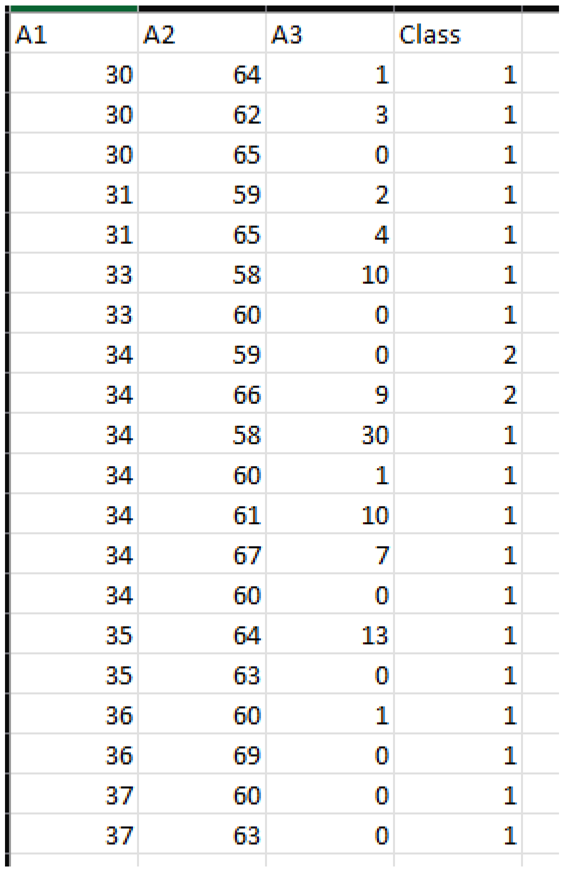
Decision Tree Induction Implementation

Generating the Decision-Tree

In our implementation, we start by splitting the data 70 to 30 for the training and testing set respectively. After that, with the full training set and root node, we call a function called generate. The input parameters are the dataset, the list of attributes, the attribute selection method, and a position indicator. This function first creates a node and checks to see if the list of attributes is empty or if the number of unique values is 1. If either of these cases is true, then the class with the majority outcome is automatically chosen as a leaf node. Next, whichever attribute selection method was specified is used to get the splitting attribute and splitting point(s). These attribute selection methods use the equations we established in the Introduction to obtain these decisions. The selected attribute is then removed from the list of attributes. For each partition the generate function is recursively called until leaf nodes are reached (due to lack of attributes or lack of attribute values to partition over). Finally, the test set is used to measure the accuracy (and other metrics) of the newly created tree.

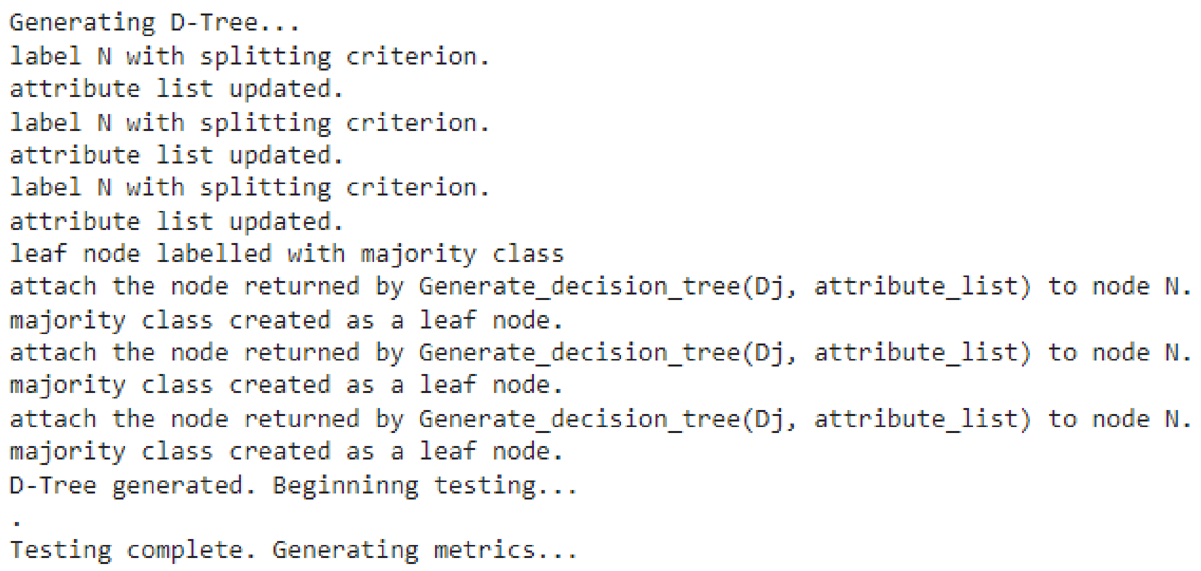
Verifying the Implementation

The *Python 3.9* implementation has been tested with a test dataset shown in figure 3. In order to verify our code, we use print statements throughout the algorithm, these prints are shown in figure 4. We further verified the implementation by comparing it with SKLearn DecisionTreeClassifier() function and visualization of the tree which is shown in figure 5.



A1	A2	A3	Class
30	64	1	1
30	62	3	1
30	65	0	1
31	59	2	1
31	65	4	1
33	58	10	1
33	60	0	1
34	59	0	2
34	66	9	2
34	58	30	1
34	60	1	1
34	61	10	1
34	67	7	1
34	60	0	1
35	64	13	1
35	63	0	1
36	60	1	1
36	69	0	1
37	60	0	1
37	63	0	1

Figure 2: Test Dataset



```
Generating D-Tree...
label N with splitting criterion.
attribute list updated.
label N with splitting criterion.
attribute list updated.
label N with splitting criterion.
attribute list updated.
leaf node labelled with majority class
attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N.
majority class created as a leaf node.
attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N.
majority class created as a leaf node.
attach the node returned by Generate_decision_tree(Dj, attribute_list) to node N.
majority class created as a leaf node.
D-Tree generated. Beginninng testing...
.
Testing complete. Generating metrics...
```

Figure 3: Print statements in implementation of Decision Tree Induction Algorithm

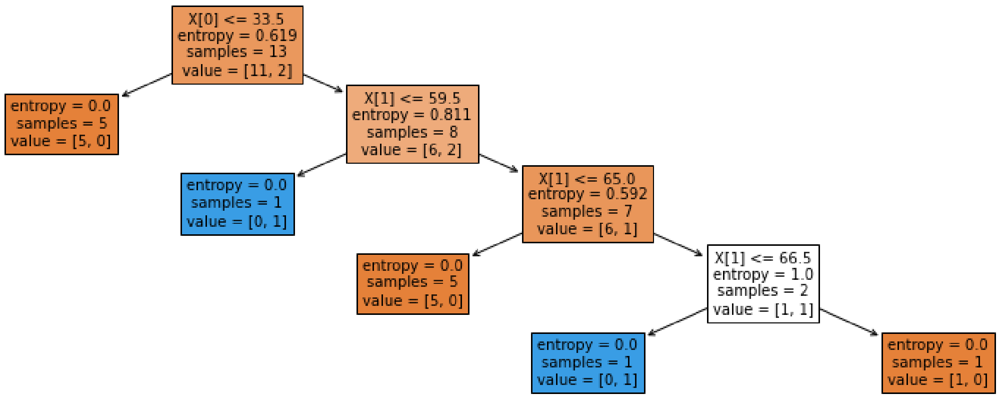


Figure 4: Decision Tree from SKLearn for comparison/verification purposes

<u>Experimental Set-Up</u>

Datasets

A total of 6 datasets have been used from the *UC Irvine Machine Learning Repository* [2], varying in size -Number of Instances-and dimension -Number of Attributes, however, they are all of an integer data type. As depicted in Figures 5 and 6, "*Haberman's Surviva*" dataset has the largest number of instances, while "*Lung Cancer*" has the largest number of attributes.

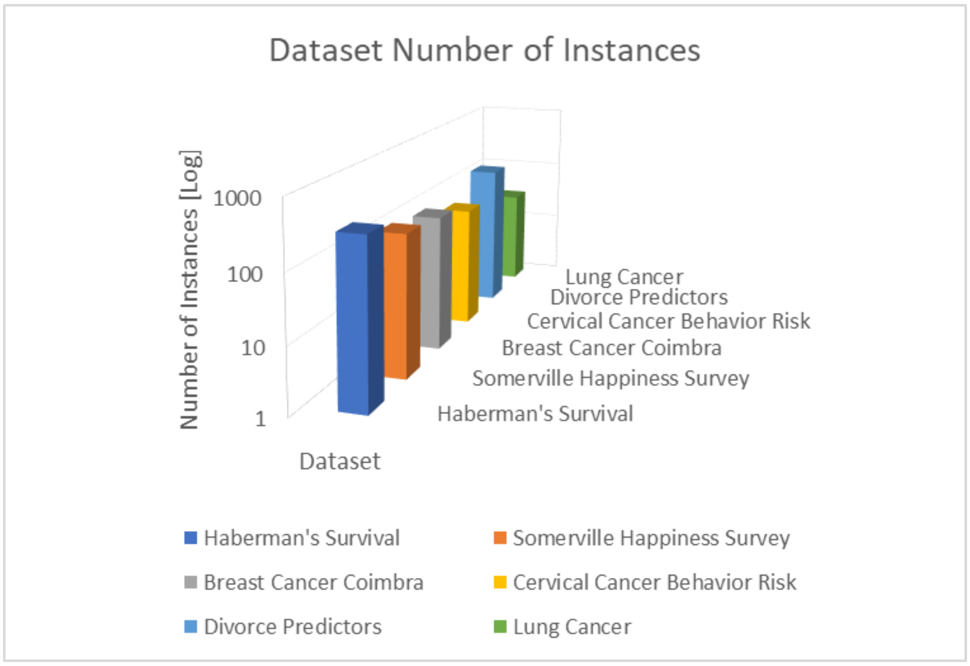


Figure 5: Plot of Datasets VS Number of Instances

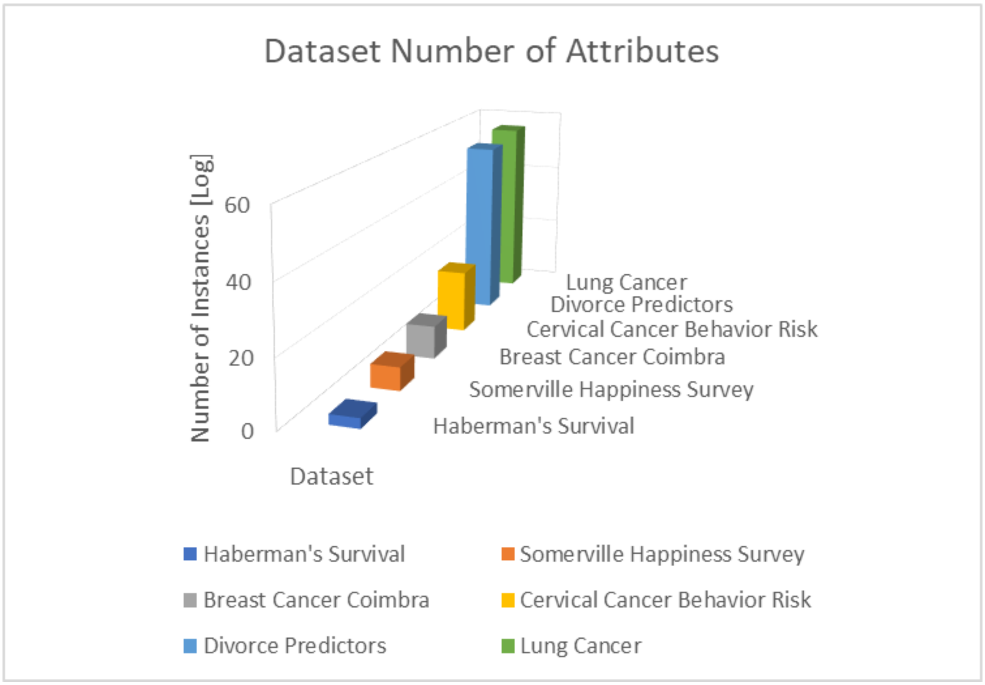


Figure 6: Plot of Datasets VS Number of Attributes

Table 1 summarizes the mentioned dataset metadata -Number of Instances, Number of Attributes, and whether it contains missing values- numerically.

	Dataset Name	Number of Instances	Number of Attributes
1	Haberman's Survival	306	3
2	Divorce Predictors	170	54
3	Somerville Happiness Survey	143	7
4	Breast Cancer Coimbra	116	10
5	Cervical Cancer Behavior Risk	72	19
6	Lung Cancer	32	56

Table 1: Dataset Details Summary Table

Preprocessing Steps

The following steps have been carried out in order to get the data ready to be inputted into the Decision Tree Induction algorithm:

I. Missing Value Treatment: Inserting ‘0’ values for the missing data points suitable to the dataset
II. Enumerating the Categorical Classes: Replacing string class labels with integer numbers

Metrics to be collected

For evaluation purposes the following utility metrics were collected:

- Execution Time
- Memory Resident Set Size (RSS) : the amount of memory being allocated to the process from RAM
- Virtual Memory Size (VMS): the entire memory space that can be accessed by the process including swap and shared memory
- Shared Memory Size
- Precision/Positive Predictive Value (PPV) = TP / TP+FP
- Sensitivity/True Positive Rate (TPR) = TP / TP+FN
- Specificity/True Negative Rate (TNR) = TN / TN +FP
- Accuracy = T / T + F

This has been achieved using *psutil 5.8.1* [3] library, a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in *Python*.

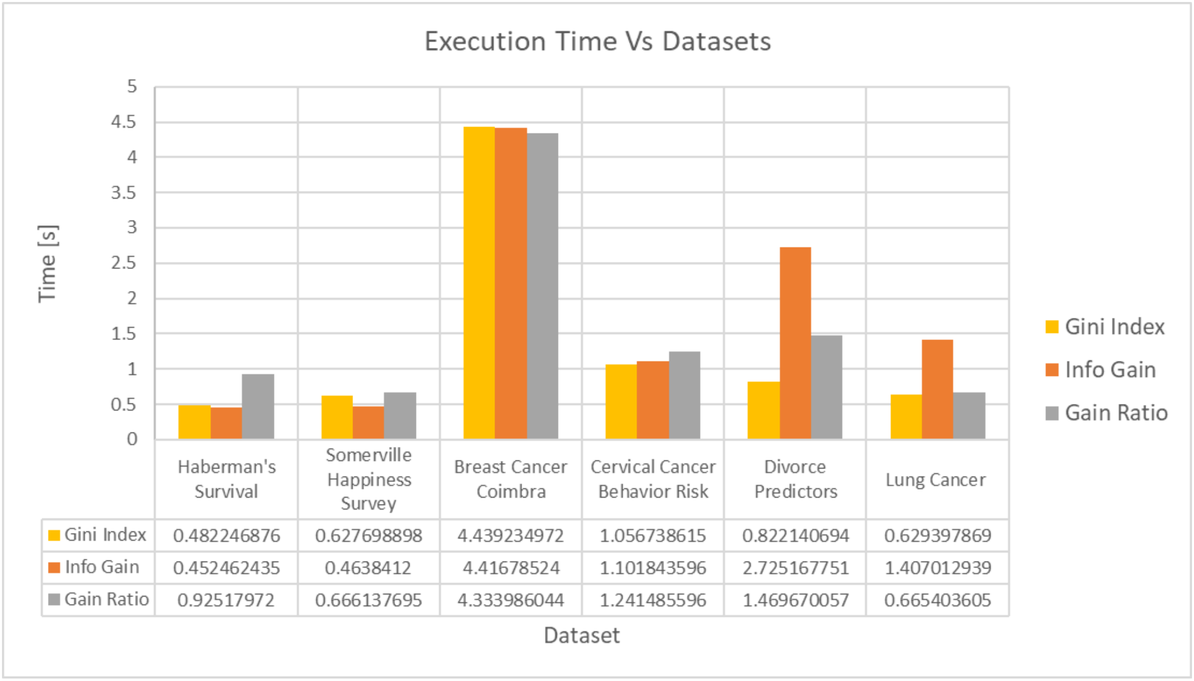
Abbreviations: T = TN +TP, F = FN +FP, T: True, F: False, N: Negative, P: Positive

Results & Discussion

Execution Time VS Datasets

The execution time has been measured for all the datasets and results are shown in figure 7. "*Breast Cancer Coimbra*" has the longest execution time, 4.44 seconds. which is almost 4 times larger than all 5 other datasets. Attribute selection methods have little to no contribution towards execution time except for "*Divorce Predictors*" and "*Lung Cancer*", where Info Gain has a significantly longer computation time. Interestingly, these two datasets have the largest dimension values of 56 and 54.

The obtained results do not match our theoretical and empirical (based on previous projects results) expectations for larger datasets to take longer in computation time. To the contrary, the largest dataset in size, "*Haberman's Survival*", has the smallest execution time. This could be due to the difference between data mining and classification tasks. Moreover, difficulty in classification in cases may be irrelevant to the size and dimension of the dataset.



	Haberman's Survival	Somerville Happiness Survey	Breast Cancer Coimbra	Cervical Cancer Behavior Risk	Divorce Predictors	Lung Cancer
Gini Index	0.482246876	0.627698898	4.439234972	1.056738615	0.822140694	0.629397869
Info Gain	0.452462435	0.4638412	4.41678524	1.101843596	2.725167751	1.407012939
Gain Ratio	0.92517972	0.666137695	4.333986044	1.241485596	1.469670057	0.665403605

Figure 7: Execution Time VS Dataset for Decision Tree Induction

Memory Resident Set Size VS Datasets

Figure 8 shows Memory Resident Set Size (RSS) usage across the datasets. The results indicate that Gain Ratio uses relatively larger RSS, except for "*Cervical Cancer Behavior Risk*" and "*Lung Cancer*" which also are the two smallest datasets in size. In addition, in these datasets Gini Index uses the largest RSS. The expectation for larger datasets in size and dimensions to have higher RSS usage is met.

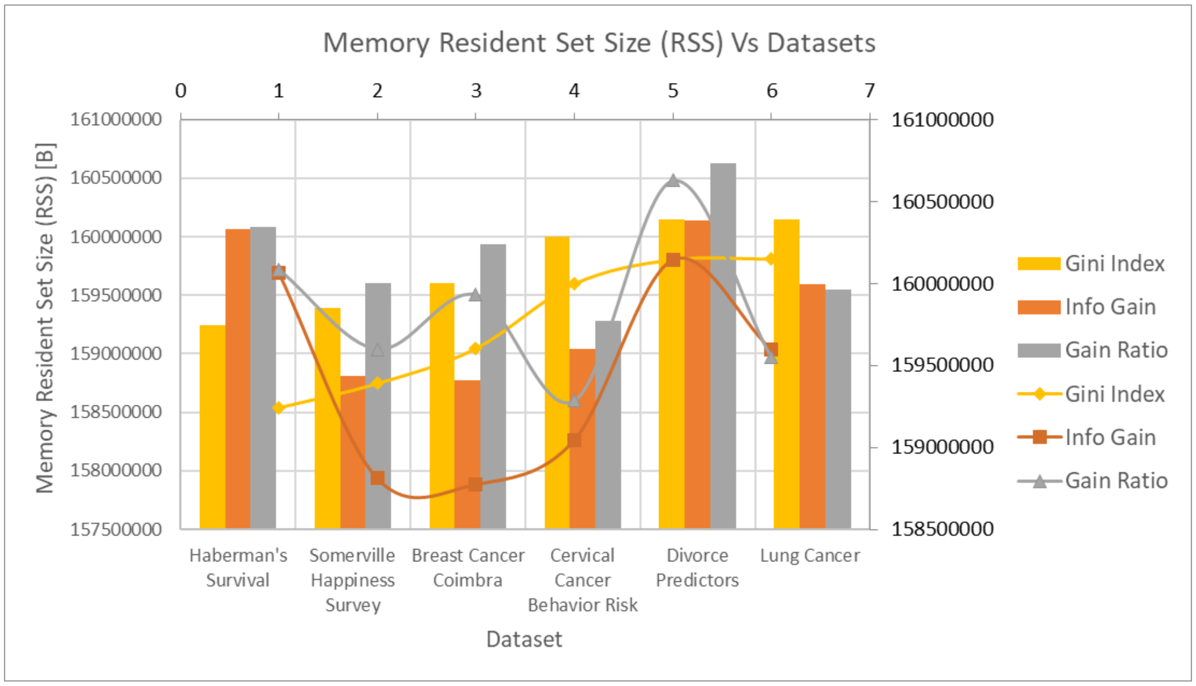


Figure 8: Memory Resident Set Size VS Dataset for Decision Tree Induction

Virtual Memory & Shared Memory Size VS Threshold

In contrast with results for data mining algorithms, Star Cubing and FP-Growth, Decision Tree Induction results of Virtual Memory (VMS) and Shared memory do not follow the similar trend. Virtual Memory results shown in figure 9 meets the expectations for a single process implementation theory. One can note that "*Divorce Predictors*" which has a combination of large size and large dimensions has the biggest use of VMS.

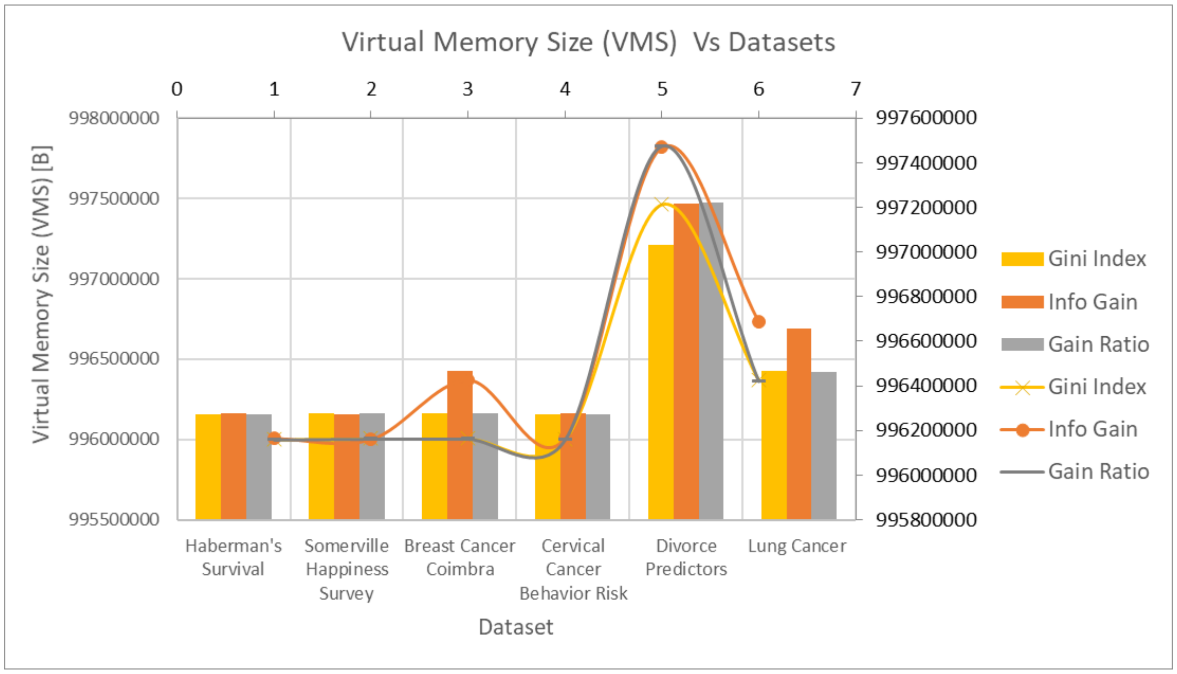


Figure 9: Virtual Memory Size VS Dataset for Decision Tree Induction

Shared Memory results are depicted in figure 10, meeting the expectations for the largest dataset in size, "*Haberman's Survival*", to have the largest shared memory. However, apart from "*Haberman's Survival*", in all other datasets, Gini Index uses more shared memory compared to Info Gain and Gain Ratio.

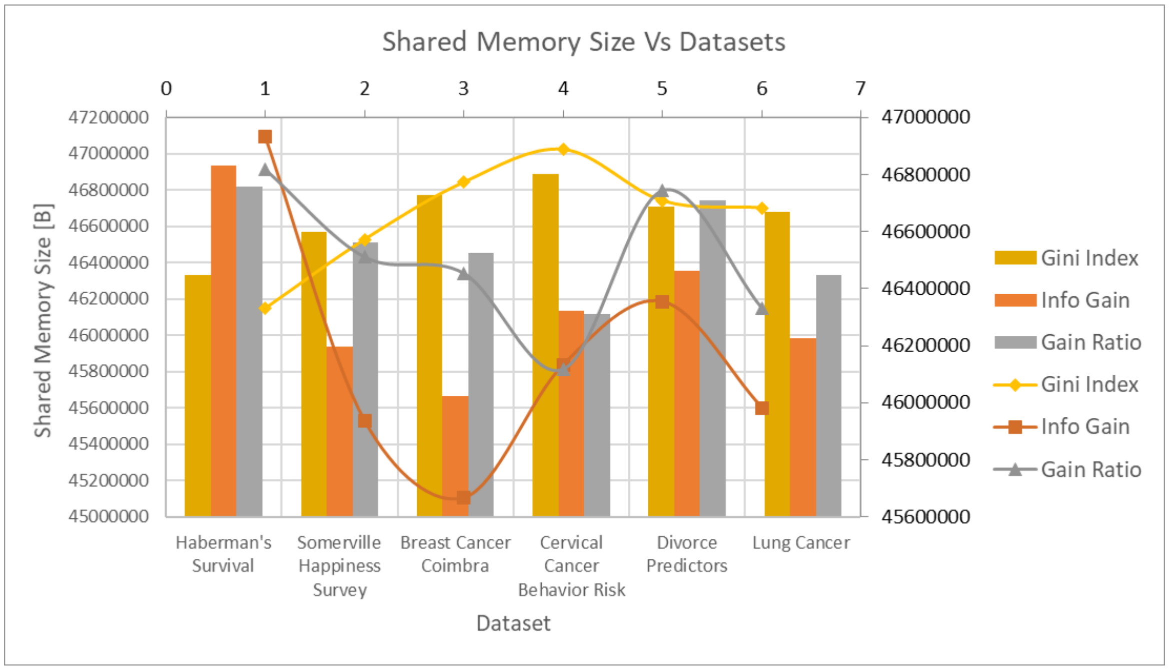


Figure 10: Shared Memory Size VS Dataset for Decision Tree Induction

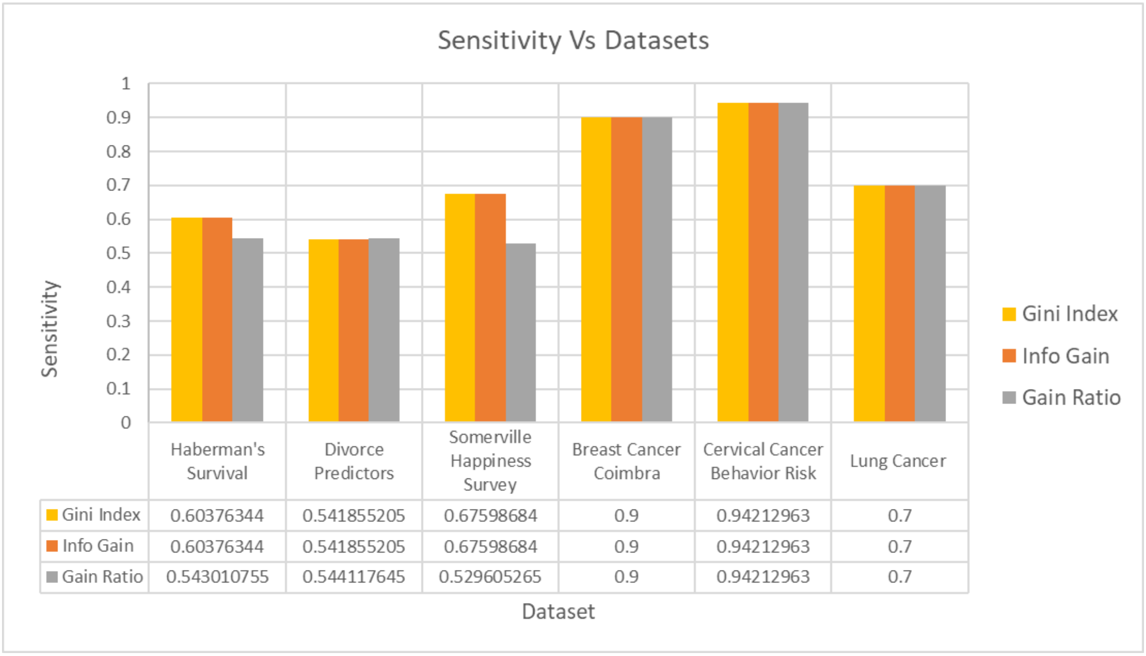
The classification metrics for sensitivity, precision, specificity and accuracy are shown in figures 11 to 14. As shown in figure 11, across all datasets all three attribute selection methods lead to the same value of sensitivity, apart from "*Haberman's Survival*" and "*Breast Cancer Coimbra*". The mentioned datasets' Gain Ratio performs lower.

The best two performing datasets are "*Divorce Predictors*" and "*Cervical Cancer Behavior Risk*".

Precision shows a very similar trend where "*Divorce Predictors*" and "*Cervical Cancer Behavior Risk*" are still the two best performing, and "*Breast Cancer Coimbra*"'s Gain Ratio is the lowest among all.

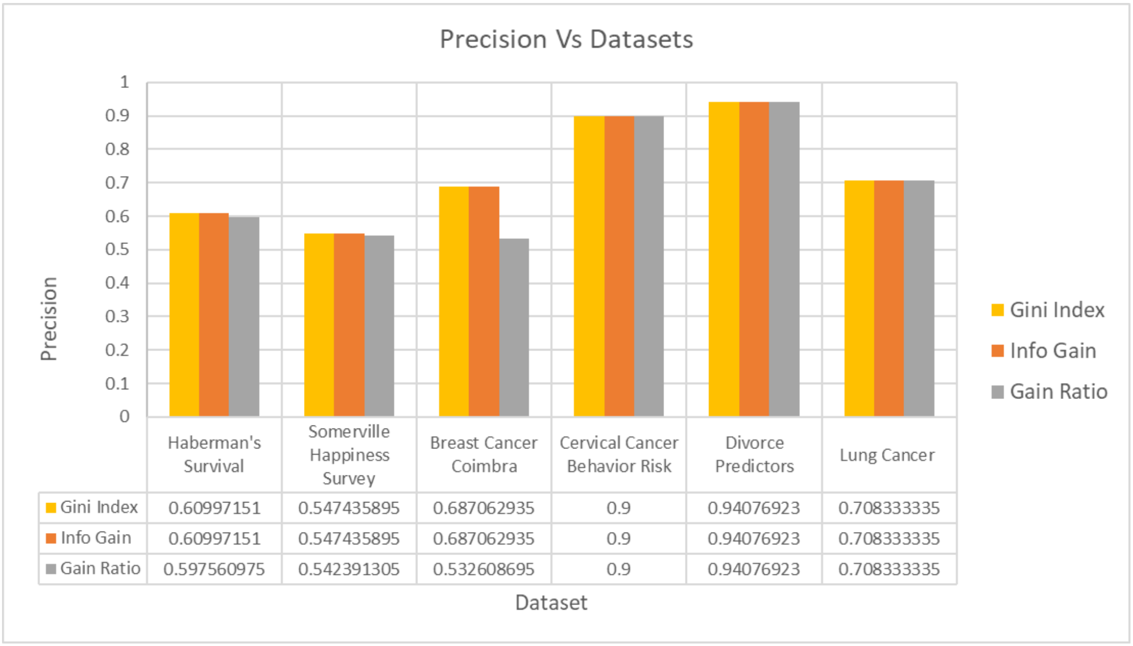
Specificity, shown in figure 13, and sensitivity results are identical.

Accuracy, shown in figure 14, also follows the general trend shown in figures 11 to 13, even though Gain Ratio behaviour is slightly different in "*Haberman's Survival*" and "*Somerville Happiness Survey*" which are 1st and 3rd largest datasets in size.



	Haberman's Survival	Divorce Predictors	Somerville Happiness Survey	Breast Cancer Coimbra	Cervical Cancer Behavior Risk	Lung Cancer
Gini Index	0.60376344	0.541855205	0.67598684	0.9	0.94212963	0.7
Info Gain	0.60376344	0.541855205	0.67598684	0.9	0.94212963	0.7
Gain Ratio	0.543010755	0.544117645	0.529605265	0.9	0.94212963	0.7

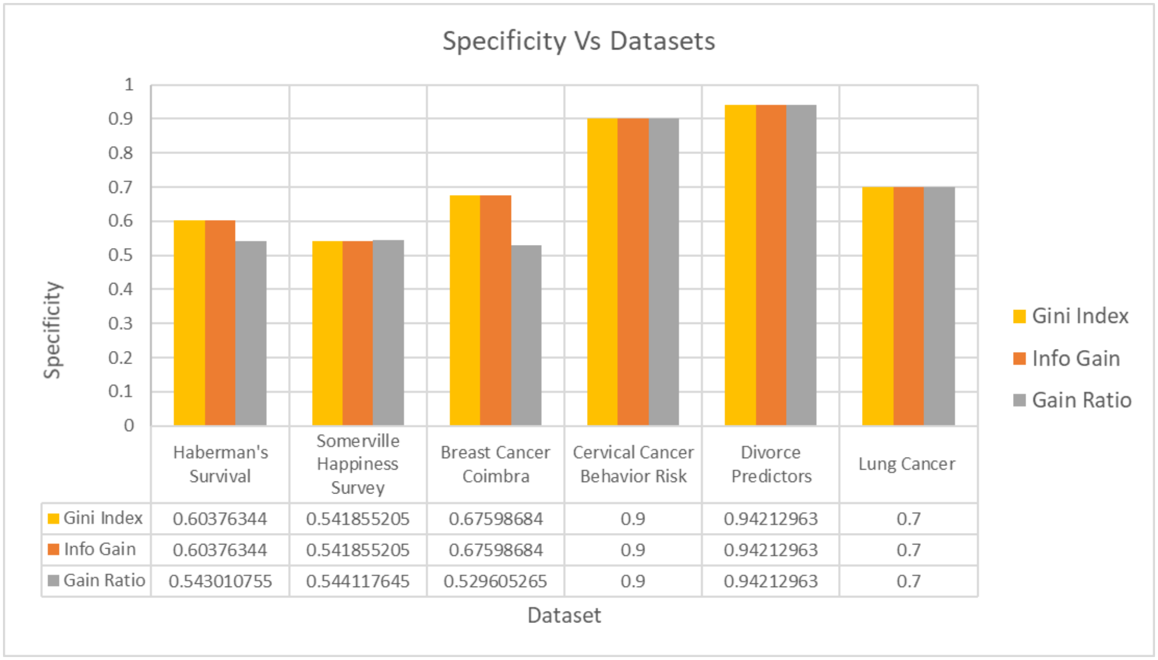
Figure 11: Sensitivity VS Dataset for Decision Tree Induction



	Haberman's Survival	Somerville Happiness Survey	Breast Cancer Coimbra	Cervical Cancer Behavior Risk	Divorce Predictors	Lung Cancer
Gini Index	0.60997151	0.547435895	0.687062935	0.9	0.94076923	0.708333335
Info Gain	0.60997151	0.547435895	0.687062935	0.9	0.94076923	0.708333335
Gain Ratio	0.597560975	0.542391305	0.532608695	0.9	0.94076923	0.708333335

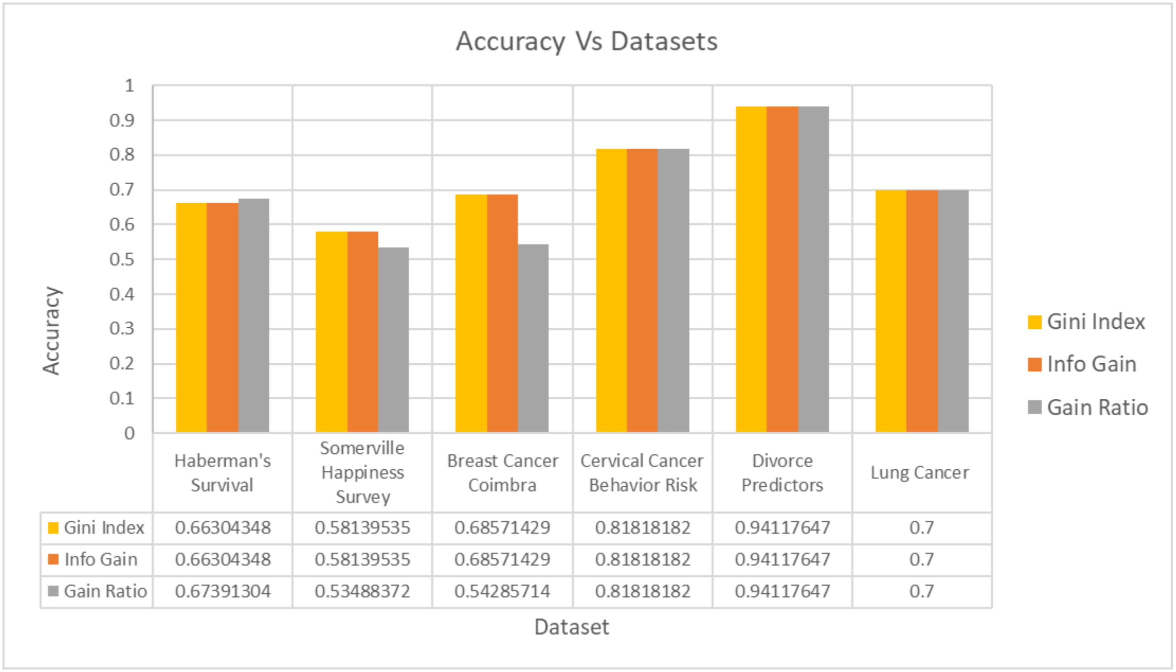
Figure 12: Precision VS Dataset for Decision Tree Induction

When comparing with all metrics and across all datasets, the two top performing datasets, "*Divorce Predictors*" and "*Cervical Cancer Behavior Risk*", have a combination of large size and high dimensions. One can argue that this combination presents the data distribution more accurately and hence provides the models with a better data representation.



	Haberman's Survival	Somerville Happiness Survey	Breast Cancer Coimbra	Cervical Cancer Behavior Risk	Divorce Predictors	Lung Cancer
Gini Index	0.60376344	0.541855205	0.67598684	0.9	0.94212963	0.7
Info Gain	0.60376344	0.541855205	0.67598684	0.9	0.94212963	0.7
Gain Ratio	0.543010755	0.544117645	0.529605265	0.9	0.94212963	0.7

Figure 13: Specificity VS Dataset for Decision Tree Induction



	Haberman's Survival	Somerville Happiness Survey	Breast Cancer Coimbra	Cervical Cancer Behavior Risk	Divorce Predictors	Lung Cancer
Gini Index	0.66304348	0.58139535	0.68571429	0.81818182	0.94117647	0.7
Info Gain	0.66304348	0.58139535	0.68571429	0.81818182	0.94117647	0.7
Gain Ratio	0.67391304	0.53488372	0.54285714	0.81818182	0.94117647	0.7

Figure 14: Accuracy VS Dataset for Decision Tree Induction

Conclusion

In this paper we implemented our own version of the Decision Tree Induction Algorithm and tested it against standard benchmarks. In discussion, we also took into account the results of previous implementations for Star Cubing and FP-Growth.

Execution time results comparison among dataset and algorithms indicates that the execution time for classification task follows a dissimilar fashion to a frequent pattern mining task; computation time is not in a direct relation with size or dimension of the dataset.

Memory measurement showed that in a general trend, Gian Ratio uses RSS and Gini Index uses shared memory.

The results for Attribute Selection methods suggest that Gain Info might be a more efficient choice of method as in a general overview, it uses relatively less memory and still performed as good as other two methods. However, the computation time for Gain Info might not be ideal.

Classification metrics were in agreement with one another and showed that a dataset with a combination of large size and high dimensions will have better performance.

References

[1]J. Han, M. Kamber, J. Pei. (2012). Data Cube Technology, *Data Mining: Concepts and Techniques* (3rd ed., pp. 187-242). Waltham, MA: Morgan Kaufmann Publishers.

[2]"UCI Machine Learning Repository", *Archive.ics.uci.edu*. [Online]. Available: https://archive.ics.uci.edu/ml/index.php. [Accessed: 14- Oct- 2021]

[3]"psutil documentation — psutil 5.8.1 documentation", *Psutil.readthedocs.io*, 2021. [Online]. Available: https://psutil.readthedocs.io/en/latest/#. [Accessed: 14- Oct- 2021]