# SCC.461 Programming for Data Scientists
# Final Project

Rawan Abdulelsadig - 35324987

January 2020

**Abstract**

In this report, the work of implementing a classification decision tree from scratch is described and its performance is statistically compared to python's sklearn library implementation "DesicionTreeClassifier", the comparison was done by trying to classify cancer as malignant or benign using the breast cancer wisconsin data-set, and evaluating the performance of the models. It was found that both implementations produced almost the same accuracy, F1-score, precision and recall as the minimum number of leaf samples required for a split changed, while in terms of fitting and prediction time, the implemented model failed to beat sklearn's optimized computational time. In contrast, the implemented model was able to be almost half the size of sklearn's model in all times (in Bytes).

## 1 Introduction

Data science is the process of extracting information from data in such a way that allows for better decisions, which are made based on better predictions. Machine learning is the trending method of building mathematical or logical models that can represent the data well enough to produce reliable inference.

Decision tree is the simplest and mostly used machine learning model that is based simply on constructing a decision boundary built using a sequence of logical conditions, this sequence of conditions separate the data into two sections in each level forming a binary tree. This algorithm can be used for regression or classification, some implementations allow a decision tree to perform both (classification and regression tree CART).

In this work, a classification decision tree was implemented (in python), fitted to the breast cancer wisconsin data-set [1, 2, 3, 5] and evaluated against a library implementation (sklearn's DecisionTreeClassifier) using various performance measures and statistical methods (in R).

In the rest of this report, the methodology taken to implement the decision tree and the different parameters that defined it will be described in section 2, along with a brief description of the classification procedure and the different evaluation metrics. The statistical tools used to compare the implementation against the library's implementation are also described in this section. In section 3, the results of comparing the two models are shown, while in section 4, those results are discussed and possible improvements are suggested.

## 2 Methods

### 2.1 Building the Decision Tree Model

Figure 1 shows a simple decision tree classifier, it can be observed that there are two basic building blocks: interior nodes (colored yellow) and exterior nodes (colored red). The interior nodes are considered as the main tree nodes that each carry a condition (operating on one feature) which splits the data into: left sub-tree and right sub-tree, while the exterior nodes are the leafs of the tree which carry the relevant information that allows for predicting; in

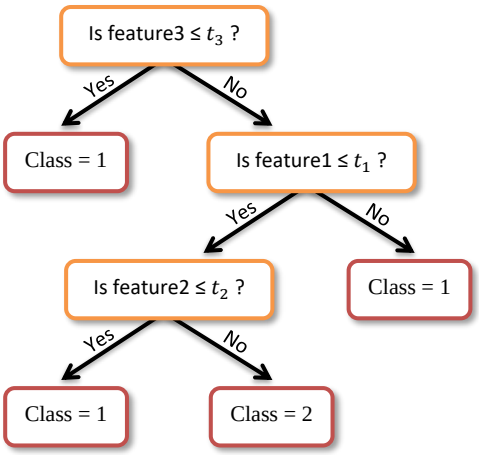



Figure 1: A Decision Tree

the case of a classification tree: the majority class of the samples in that split of the data.

One important parameter of a decision tree is the splitting criterion that identifies how a split is made in each level, generally a split is made in such a way that increases the purity (or decreases the impurity) of the child nodes. The purity of a node is related to the associate classes of the samples in that node; a node that only contains samples belonging to the same class is a perfectly pure node (impurity = 0). Examples of purity (or impurity) measures are gini index and entropy.

Gini index measures the divergence of the probability distributions of the classes in a split [4], the lower the gini index the lower the impurity.

$$Gini(D) = 1 - \sum_{i=1}^{m} \left[ \frac{|D_{class=i}|}{|D|} \right]^2 \tag{1}$$

Entropy in a general sense is a measure of disorder. And in the context of purity measuring, the higher the disorder the higher the impurity.

$$Entropy(D) = \sum_{i=1}^{m} -\frac{|D_{class=i}|}{|D|} log_2 \left( \frac{|D_{class=i}|}{|D|} \right) \tag{2}$$

In equations 1 and 2, m is the number of classes in the data and $|d|$ means the number of samples or data points in the specified set "d".

The quality of a split is then evaluated using a weighted sum of the impurity measure in both sides (left sub-set and right sub-set), the primary task is to find the value y and feature $X_i$ that produce the the lowest overall impurity identified as:

$$Impurity_{(y,X_i)}(D_{left}, D_{right}, D) = \frac{|D_{left}|}{|D|} criterion(D_{left}) + \frac{|D_{right}|}{|D|} criterion(D_{right}) \tag{3}$$

Where $D_{left}, D_{right}$ and $D$ are the left sub-set of the data, the right sub-set of the data and the whole set of the data in that level, respectively. Criterion can be gini or entropy as spesified in equations 1 and 2.

The left and right sub-sets are obtained by splitting the data using a certain feature ($X_i$) in the following manner:

$$Split(y, X_i, D) \Rightarrow \begin{cases} D_{left} & \Leftarrow D_{X_i <= y} \\ D_{right} & \Leftarrow D_{X_i > y} \end{cases}$$

Finding y and $X_i$ is done by evaluating the impurity produced by every possible split y in every feature dimension $X_i$ where i = 1,2,..., Number Of Features. Since this can be computationally expensive, another way is to randomly choose a feature $X_i$ then randomly choose a splitting value y, although this method can hurt the purity of the splits, it can significantly cut down the computational burden of fitting a decision tree.
Despite the fact that a randomized decision tree is often not an accurate classifier, having an ensemble of randomized fully grown decision trees is known to be a powerful model.

The basic framework of building the decision tree is to recursively find and perform a split until a stopping condition is reached. A stopping criterion can be the maximum depth of the tree (which is the number of tree levels) or the minimum number of samples in a leaf node, those criteria can also help in the regularization of the decision tree and the prevention of over-fitting to the data and consequently improving generalization. Information gain can also be a stopping criterion by identifying a minimum amount of information gain needed to approve a split, it can be considered as active pruning. It is calculated as follows:

$$Information\ Gain = Entropy(Parent) - WeightedSumOfEntopy(Children) \tag{4}$$

**Algorithm 1** Building The Decision Tree

**Inputs:**
Data: The data as a 2d array, with the target classes as the last column.
**Global Constants:**
Min-Split: The minimum number of samples in a node required to perform a split.
Max-Depth: The maximum number of tree levels allowed.
Min-InfoGain: The minimum amount of information gain allowed for a split to be carried out.
**Output:**
The root node of the generated tree as an abstract datatype "TreeNode".

```
1:  procedure BuildTree(Data)
2:      if |Data| ≥ Min-Split then
3:          X_i, y, D_left, D_right ← PerformSplit(Data)
4:      else
5:          return LeafNode(|Data|, ModeClassofData)
6:      if InformationGain(D_left, D_right, Data) ≤ Min-InfoGain then
7:          return LeafNode(|Data|, ModeClassofData)
8:      else
9:          if CurrentLevel < Max-Depth - 1 then
10:             LeftSubTree ← BuildTree(D_left)
11:             RightSubTree ← BuildTree(D_right)
12:         else
13:             LeftSubTree ← LeafNode(|D_left|, ModeClassofD_left)
14:             RightSubTree ← LeafNode(|D_right|, ModeClassofD_right)
15:         return TreeNode(X_i, y, |Data|, LeftSubTree, RightSubTree)
```

TreeNode and LeafNode are abstract data types that represent the yellow and red squares in Figure 1, respectively.

Predicting a new data point is done by following the conditions along the tree until reaching a LeafNode that carries a ModeClass which is the mode of the classes of the leaf samples that were used to fit the tree, this ModeClass is the predicted class of that data point.

Six main model parameters were specified:

- Min_Leaf_Samples: The minimum number of samples in a child node that is allowed to consider a split, whenever a resulting child node contains a number of samples less than this value the split will not be considered.
- Min_Leaf_Split: The minimum number of samples in a node that is required to split it, whenever a node contains a number of samples less than this value the node will stay a leaf and will not get split. This parameter must be $\geq 2*$Min_Leaf_Samples.
- Max_Depth: The maximum number of levels the tree is allowed to grow to, note that this is not necessarily the final depth of the tree.
- Criterion: The impurity measure: gini or entropy.
- Random_Split?: Whether the split is made randomly or by evaluating all possible splits "best split".
- Infogain_Threshold: An extra parameter (not implemented in sklearn) that specifies the minimum information gain that allows a split to be finalized, allowing a split that produces 0 information gain results in a fully grown un-proned over-fitted tree. This parameter works as an active pruning mechanism when set to a value greater than 0.

To obtain the results in section 3, the default value of Max_Depth was set to be 50, Min_Leaf_Samples was set to 1 and Infogain_Threshold was set to 0 (when the information gain is $\leq 0$ the split is not carried out). However Min_Leaf_Split was varied in the range [2, 170] to show how the model performs as this parameter changes. criterion and Random_Split were configured to cover four cases: Gini criterion + best splits, Gini criterion + random splits, entropy criterion + best splits and entropy criterion + random splits.

## 2.2 Classification and Evaluation Measures

Building and evaluating a classification machine learning algorithm is typically done by splitting the data to training and testing sets; the training set is used to build the model while the testing set is kept aside to simulate new unseen data that gives an estimate measure of the generalization capabilities of the model.
K-fold cross validation is sophisticated procedure where the splitting, training and evaluation is done K times, it is used to evaluate the effect of changing a parameter in the generalization performance of the model. It is done by choosing a value of the hyper-parameter, randomly shuffling and splitting the data into K equal sections or "folds", then training on K-1 folds of the data and testing on the remaining fold, this guarantees that all the data samples were used for both training and testing the model at least once. The evaluation measures are then averaged across all the K-fold-cross-validation iterations to obtain the final approximations.

Model evaluation is highly dependant on the problem the model is intended to solve, a good model in the scope of one problem may be a bad model for solving another.
Accuracy is the first evaluation metric that comes to mind when facing a classification task, it is defined as:

$$Accuracy = \frac{Number\ Of\ Correctly\ Classified\ Instances}{Number\ Of\ All\ Instances} \tag{5}$$

However, accuracy can be an inaccurate measure when dealing with classification tasks that involve imbalanced classes (highly skewed proportions of classes), therefore, other measures are also used for evaluation such as: precision, recall and F1-Score.

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives} \tag{6}$$

$$Recall = \frac{True\ Positives}{True\ Positives\ +\ False\ Negatives} \tag{7}$$

$$F1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{8}$$

In the context of binary classification, precision is the number of positive instances that are correctly classified out of all predicted positive instances, while recall is the number of positive instances correctly classified out of all truly positive instances. F1-score is the harmonic mean of precision and recall.

Other model evaluation measures are the computation time spent on fitting the model, the computation time spent on predicting new instances and the amount of memory the model occupies.

### 2.3 Statistical Evaluation Against Library Implementation

Comparing two models is a matter of judgment, however judgment is subjective and inconclusive, using statistical tools can provide an objective view of the differences between the models and whether the difference is significant or not.
T-test is a way of examining the significance of the difference between the models' evaluation metrics as a shared parameter changes. For instance in this work, the minimum number of node samples required to perform a split was used to evaluate the two models by changing its value and recording the metrics each time. This type of t-test is known as a paired t-test; where the value of the difference of the evaluation measure between the two models is considered to be independant and identically distributed (i.i.d), and this difference can be tested for significance by setting the null hypothesis to be: the mean difference is equal to 0, while the alternative hypothesis set as: the mean difference is not equal to 0. The t-test was used in this work at 5% level by comparing the p-value obtained form the t-statistic to 0.05; a p-value $< 0.05$ indicate that the difference between the two models is significant.

$$Paired\ T\text{-}Statistic = \frac{Mean\ Of\ The\ Differences}{Standard\ Deviation\ Of\ The\ Differences/\sqrt{N}} \tag{9}$$

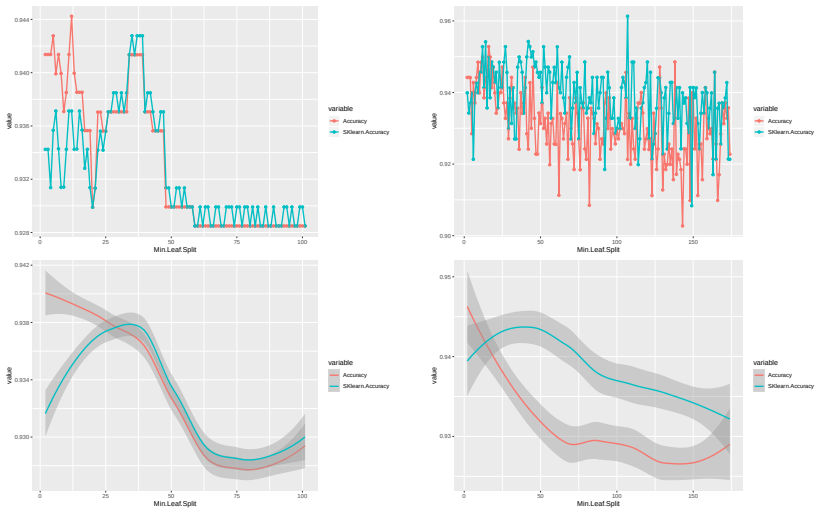
Where N is the number of evaluation trials, or in other words the length of the difference list.

## 3 Results

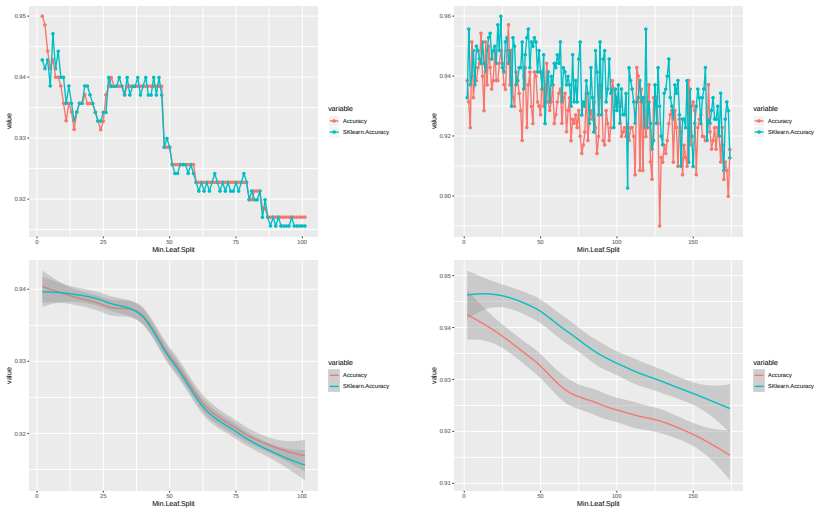
In section 2.2, seven different evaluation measures were mentioned: accuracy, precision, recall, F1-score, fitting time, prediction time and model size. All those measures along with an additional characteristic which is the tree depth were recorded for both the implemented decision tree and sklearn's DecisionTreeClassifier, those evaluation metrics were estimated using stratified 10-fold cross validation (stratification allows reserving the percentages of the positive and negative class in all folds) with the parameter Min_Leaf_split varying from 2 to 170 (it was clipped at 100 in the cases where the measures did not change significantly beyond it).

Figures 2,3,4,5,6,7,8 and 9 show the resulting plots, with the color orange referring to the developed model and the color cyan referring to sklearn's. The top sub-plots show the observed values while the bottom sub-plots show the regression lines and their 95% confidence intervals (the gray areas around the lines). The four sub-figures (a, b, c and d) show the results when using the configurations Gini criterion + best split, Gini criterion + random split, entropy criterion + best split and entropy criterion + random split, respectively.

Note: The plots in figures 2,3,4,5,6,7,8 and 9 were resized to fit the document, however they are vector-based plots that allow zooming in for a clearer view.




(a) Gini , Best (b) Gini , Random



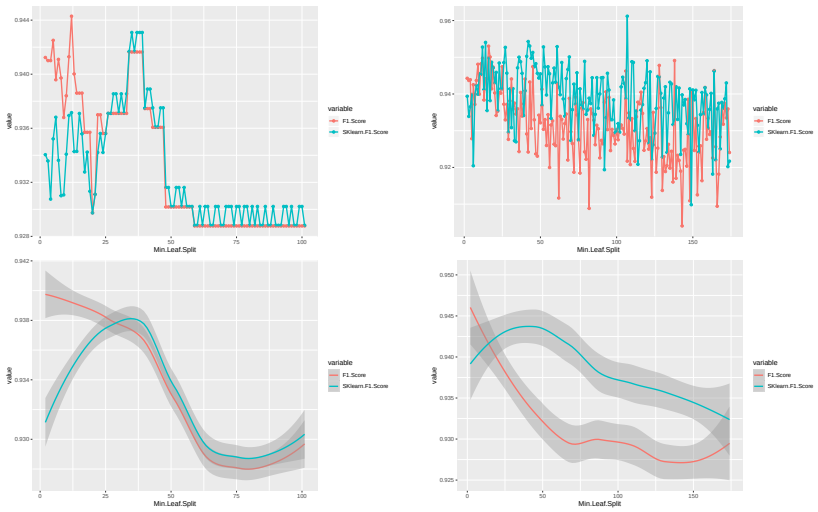

(c) Entropy , Best (d) Entropy , Random

Figure 2: Min-Leaf-Split Against **Accuracy**, Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Accuracy Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | 0.0004 | 0.0937 |
| Gini , Random | -0.0076 | 5.6795e-15 |
| Entropy , Best | 2e-04 | 0.258 |
| Entropy , Random | -0.0087 | 5.7609e-17 |

Table 1: Mean **Accuracy** Differences and Associated P-values

Figure 2 and Table 1 above show that in general; the implemented model provided slightly better accuracy when using "best" splits, however, this difference was not statistically significant. On the other hand, when random splits were used, sklearn's implementation was slightly better, and the difference was statistically significant.




(a) Gini , Best (b) Gini , Random

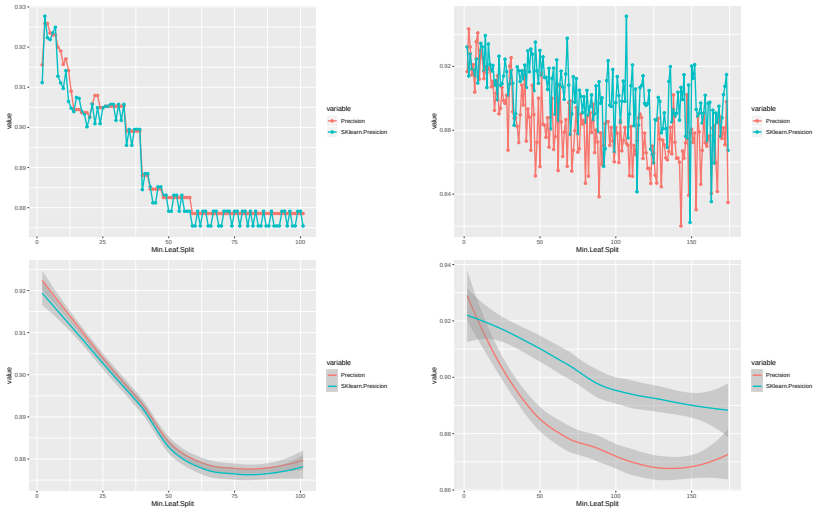
(c) Entropy , Best (d) Entropy , Random

Figure 3: Min-Leaf-Split Against **F1-Score**, Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean F1-Score Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | 0.0004 | 0.1011 |
| Gini , Random | -0.0074 | 2.5283e-14 |
| Entropy , Best | 0.0003 | 0.1128 |
| Entropy , Random | -0.0084 | 3.1351e-16 |

Table 2: Mean **F1-Score** Differences and Associated P-values

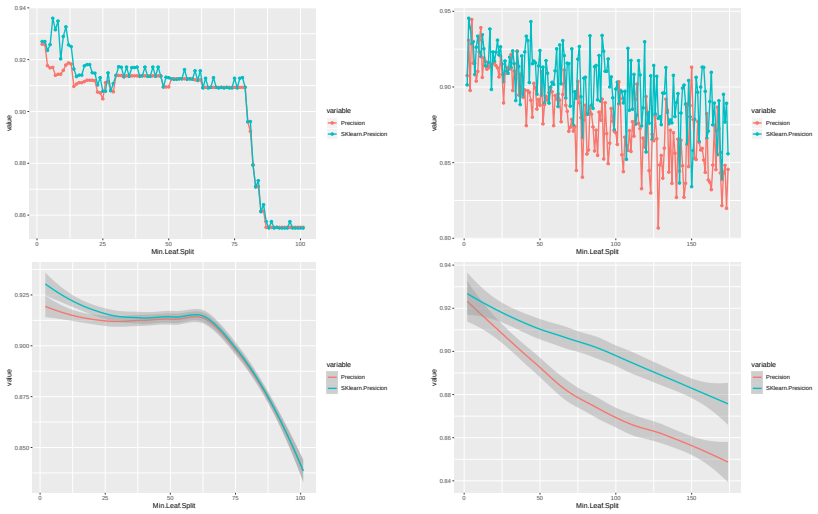
Figure 3 and Table 2 above show that again the implemented model provided slightly better results when using "best" splits and this difference was not statistically significant. On the other hand, when random splits were used, sklearn's implementation was slightly better, however, the difference was statistically significant.




(a) Gini , Best




(b) Gini , Random




(c) Entropy , Best



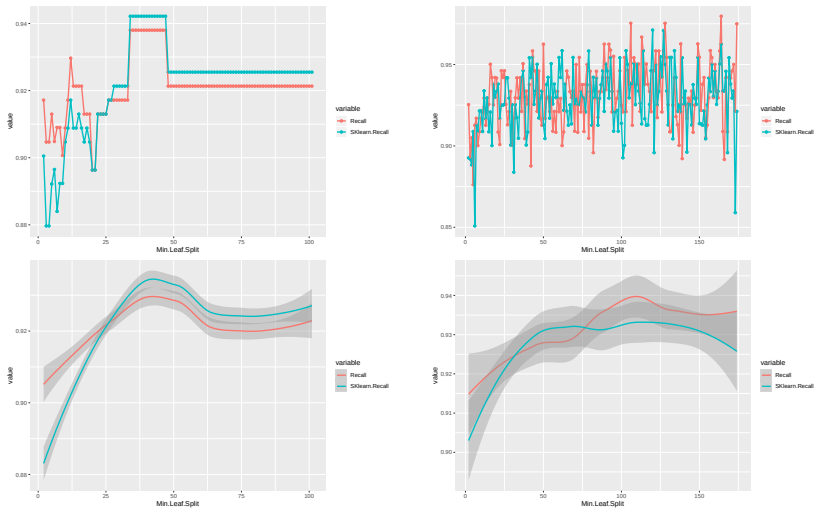

(d) Entropy , Random

Figure 4: Min-Leaf-Split Against **Precision**, Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Precision Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | 0.0015 | 1.362e-08 |
| Gini , Random | -0.0205 | 2.3509e-23 |
| Entropy , Best | -0.0025 | 2.0665e-08 |
| Entropy , Random | -0.0215 | 2.1003e-22 |

Table 3: Mean **Precision** Differences and Associated P-values

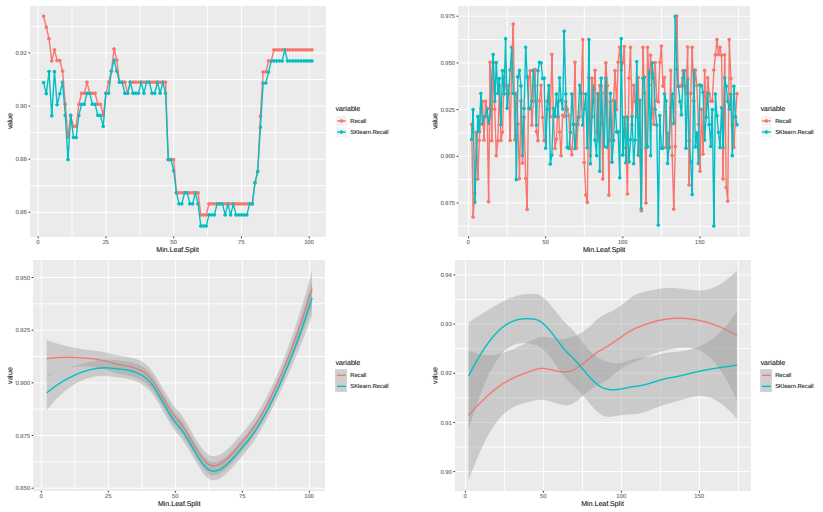
Figure 4 and Table 3 above show that in general sklearn's implementation provided a slightly better precision in almost all cases.




(a) Gini , Best




(b) Gini , Random




(c) Entropy , Best



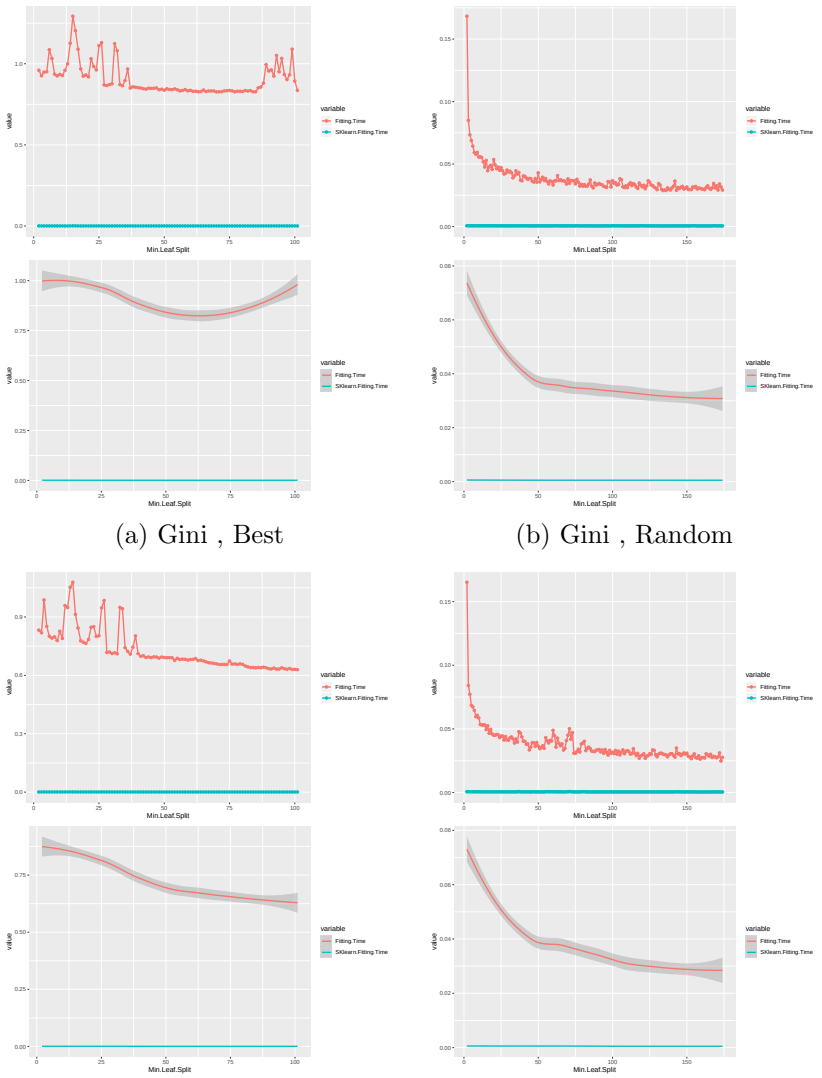

(d) Entropy , Random

Figure 5: Min-Leaf-Split Against **Recall**, Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Recall Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | -0.0007 | 0.3296 |
| Gini , Random | 0.0028 | 0.1254 |
| Entropy , Best | 0.0039 | 9.1709e-14 |
| Entropy , Random | 0.0016 | 0.4789 |

Table 4: Mean **Recall** Differences and Associated P-values

Figure 5 and Table 4 show that the implemented model and sklearn's implementation provided almost the same recall scores (the difference is not statistically significant). When using the entropy criterion along with "best" splits; the difference was statistically significant, however, it was still a small difference.




(a) Gini , Best

(b) Gini , Random

(c) Entropy , Best

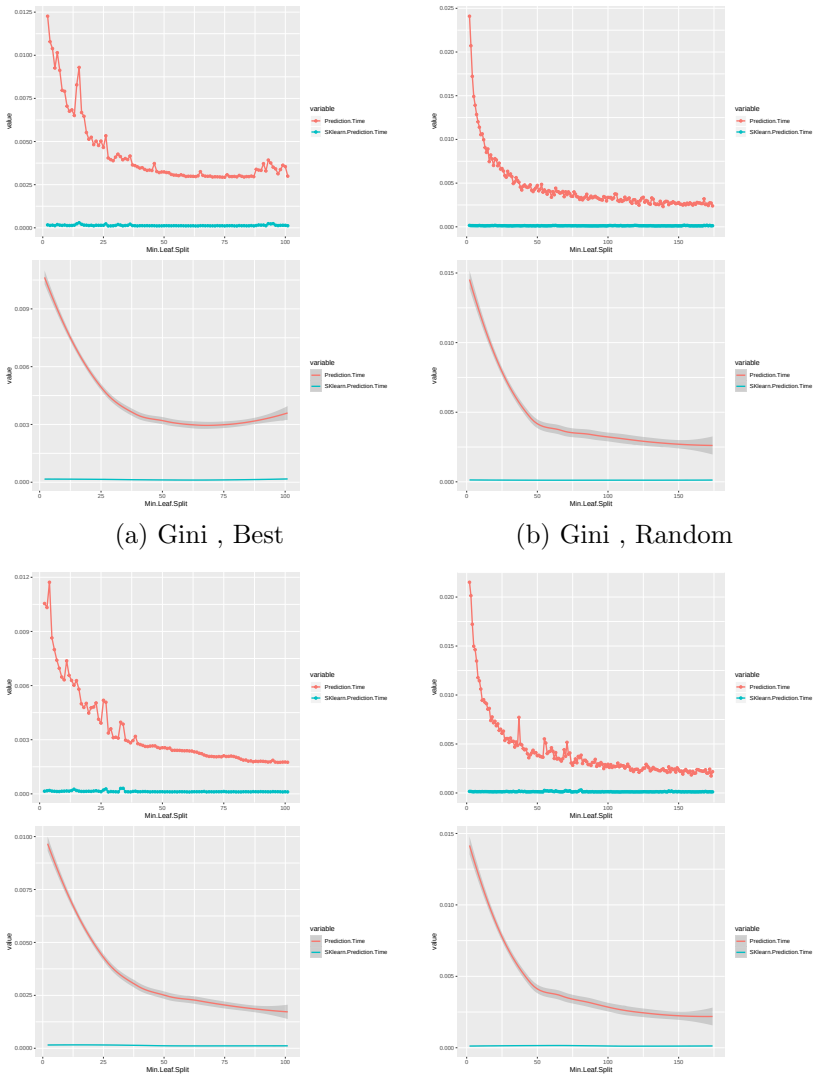
(d) Entropy , Random

Figure 6: Min-Leaf-Split Against **Fitting Time** (in seconds), Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Fitting Time Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | 0.9054 s | 2.9226e-98 |
| Gini , Random | 0.0374 s | 2.5052e-84 |
| Entropy , Best | 0.7269 s | 2.8423e-86 |
| Entropy , Random | 0.0369 s | 1.0336e-80 |

Table 5: Mean **Fitting Time** Differences and Associated P-values

Figure 6 and Table 5 show that the fitting time of sklearn's implementation was significantly and consistently less than the implemented model's, especially when using "best" splits.




(a) Gini , Best

(b) Gini , Random

(c) Entropy , Best

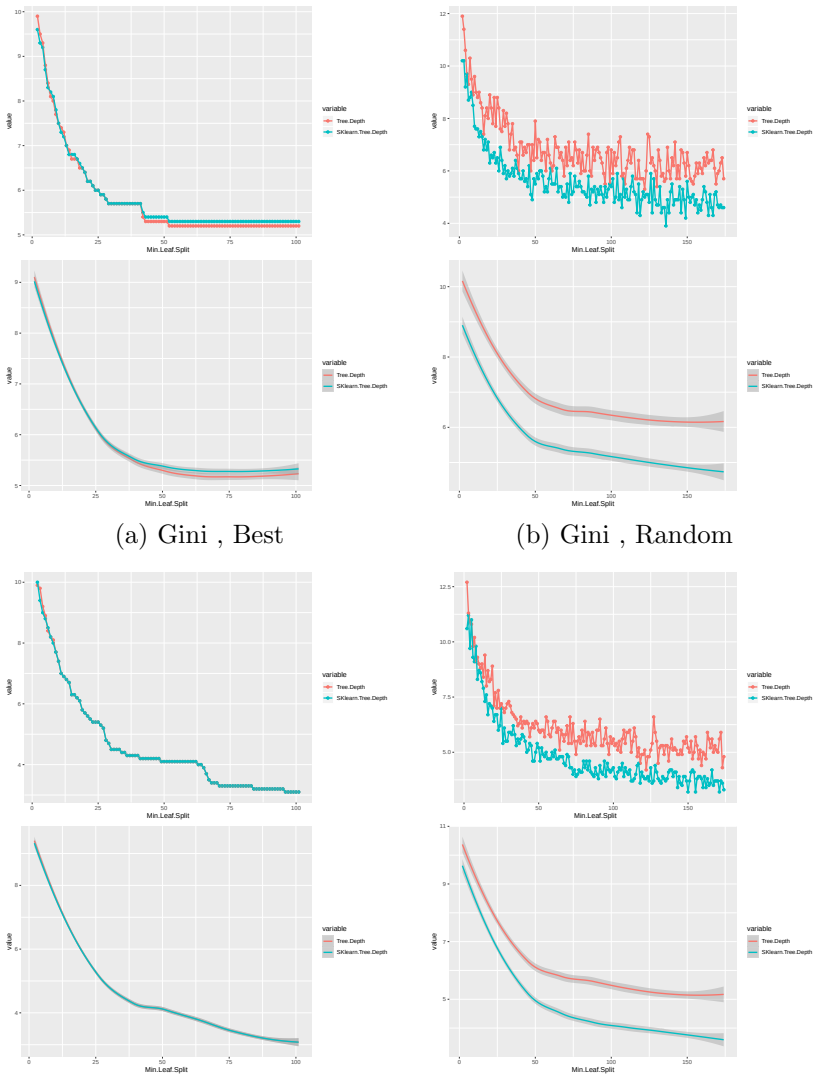
(d) Entropy , Random

Figure 7: Min-Leaf-Split Against **Prediction Time** (in seconds), Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Prediction Time Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | 0.0041 s | 3.2569e-37 |
| Gini , Random | 0.0044 s | 2.4717e-42 |
| Entropy , Best | 0.0033 s | 1.8433e-28 |
| Entropy , Random | 0.0041 s | 1.2048e-386 |

Table 6: Mean **Prediction Time** Differences and Associated P-values

Figure 7 show that sklearn's prediction time was consistently low in all cases (between 0.1 and 0.15 ms) while the implemented model's prediction time was high whenever Min_leaf_Split $< 25$ in the case of using best splits and $<$ 50 when using random splits. Table 6 shows that the prediction time of sklearn's implementation was significantly and consistently less than the implemented model's (almost zero). However, the difference was almost the same in all cases (around 40 ms).




(a) Gini , Best

(b) Gini , Random

(c) Entropy , Best

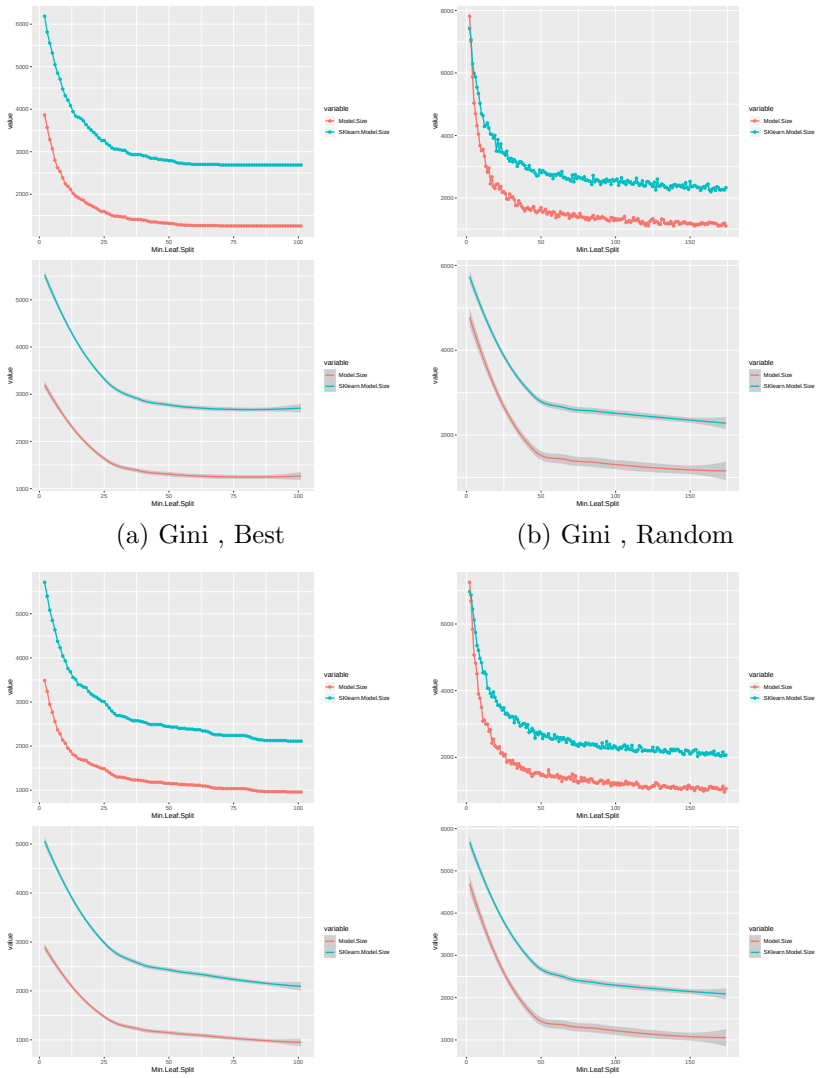
(d) Entropy , Random

Figure 8: Min-Leaf-Split Against **Tree Depth**, Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Tree Depth Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | -0.055 | 4.6098e-11 |
| Gini , Random | 1.2399 | 1.0177e-64 |
| Entropy , Best | 0.006 | 0.2224 |
| Entropy , Random | 1.2405 | 5.0073e-697 |

Table 7: Mean **Tree Depth** Differences and Associated P-values

Figure 8 and Table 7 show that the tree depth of sklearn's implementation was almost consistently less by one level when using random splits, while in the case of using best splits the resulting tree depth was approximately the same in both models.




(a) Gini , Best

(b) Gini , Random

(c) Entropy , Best

(d) Entropy , Random

Figure 9: Min-Leaf-Split Against **Model Size** (in Bytes), Criterion Set as Gini or Entropy and Spliting Method Set as Random or Best Split.

| **Configuration** | **Mean Model Size Difference** (implementedModel's - sklearnModel's) | **P-value** |
|---|---|---|
| Gini, Best | -1588.761 Bytes | 2.5335e-83 |
| Gini , Random | -1198.958 Bytes | 6.3747e-137 |
| Entropy , Best | -1384.715 Bytes | 1.1741e-73 |
| Entropy , Random | -1124.116 Bytes | 3.3e-136 |

Table 8: Mean **Model Size** Differences and Associated P-values

Figure 9 and Table 8 show that the model size of sklearn's was almost consistently heavier by more than 1 KB compared to the implemented model.

# 4 Discussion

In section 3 the detailed results obtained when comparing the sklearn's DecisionTreeClassifier and the implemented decision tree were shown, although the results show the statistical significance of some of the differences of the metrics we can still see that some differences are not quantitatively significant especially when looking at the accuracy, F1-score and recall.
However, the difference in precision can be quantitatively noticeable when using random splits where the sklearn model gave an average of 2.05% and 2.15% more precision when using gini and entropy respectively (refer to table 3).

On the other hand, the mean differences of fitting time was both statistically and quantitatively significant; fitting sklearn's model was $\approx 37$ ms faster than the implemented model when performing random splits, and $> 600$ ms when performing best splits (refer to figure 6).

Prediction time was a also statistically and quantitatively significant; in general (referring to figure 7), the sklearn's prediction time was consistently low in all cases while the implemented model consumed more time when Min_Leaf_Split was small (Min_Leaf_Split $<\approx 50$ when random splitting was used, and Min_Leaf_Split $<\approx 25$ when best splitting was used) with a maximum difference of $\approx$ 10-20 ms when Min_Leaf_Split = 2, greater values of Min_Leaf_Split resulted the prediction time of the implemented model to remain almost constant. Although, from a statistical point of view; it can be observed that the difference was depending on the value of the shared parameter, which violates the t-test assumption of having identically distributed differences, however, the test still suggested the statistical significance which aligns with the immediate observation

Both models were found to produce the same depth of trees when best splits were used, however, when random splits were used the implemented model tended to produce trees that are deeper by one level in average. In the case of model size, the implemented model was consistently $\approx$ 1-1.5 KB less than sklearn's due to the choice of only saving the number of samples in a leaf and the mode class instead of saving the actual data samples.

Since sklearn's fitting and prediction time was clearly optimized to be consistently low no matter how deep the tree, the implemented model was less efficient from this aspect which requires further tweaking of the fitting and predicting procedures; one way to significantly reduce the computational time is to build and traverse the two sides of the tree in parallel instead of the sequential method implemented in this work. However, recursively computing the two sides in parallel can impose some great complications that need great care.

# 5 Acknowledgements

The breast cancer databases was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.

# References

[1] Kristin P Bennett and Olvi L Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization methods and software*, 1(1):23–34, 1992.

[2] Olvi L Mangasarian, R Setiono, and WH Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization*, pages 22–31, 1990.

[3] Olvi L Mangasarian and William H Wolberg. Cancer diagnosis via linear programming. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1990.

[4] Lior Rokach and Oded Maimon. *Decision Trees*, volume 6, pages 165–192. 01 2005.

[5] William H Wolberg and Olvi L Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23):9193–9196, 1990.