SCC.461 Report

36827814

2025-01-19

Abstract

This report examines the finds of the usage of two Decision Tree Classifiers on three sets of data. It will provide analysis in two areas. The first area focuses on the computational aspects that went into training these Decision Tree Classifiers and the second area, analyses the machine learning aspects themselves before forming some conclusions about the classifier and data from these two separate areas.

Introduction

This report examines data generated by two Decision Tree Classifiers. The first Decision Tree Classifier was created and implemented by us and the second Decision Tree Classifier was implemented from the SKLearn.tree library. The results of which are compared later during the Results section. The first decision tree (referred to as DT-One) was created on the system specifications listed below:

```
System: Windows
Release: 10
Version: 10.0.19045
Machine: AMD64
Processor: Intel64 Family 6 Model 94 Stepping 3, GenuineIntel

CPU Information:
Processor: Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
Physical Cores: 4
Logical Cores: 4

Memory Information:
Total Memory: 8467763200 bytes
Available Memory: 1055424512 bytes
Used Memory: 7412432896 bytes
Memory Utilization: 87.5%
```

The second decision tree (DT-Two) was also ran on the same system. Both of the DTs were also ran on another system, the specifications are listed below:

```
System: Darwin
Release: 23.6.0
Version: Darwin Kernel Version 23.6.0: Fri Nov 15 15:13:28 PST 2024;
root:xnu-10063.141.1.702.7~1/RELEASE_X86_64
Machine: x86_64

CPU Information:
Processor: i386
Physical Cores: 6
Logical Cores: 12

Memory Information:
Total Memory: 17179869184 bytes
Available Memory: 6919766016 bytes
Used Memory: 8981368832 bytes
Memory Utilization: 59.7%
```

Both of the Decision Trees were tested on two sets of data with differing numbers of instances and variables. The first dataset was the 'Iris' dataset [1] which is a smaller dataset based on a classic dataset from Fisher [2] in which he proposed the idea of linear discriminant analysis (LDA) and applied it to classification of flower species based on measurements of petals and sepals. We choose this dataset as it considered one of the first foundational pieces of research which influenced machine learning. The Iris dataset is often cited in machine learning and decision tree classifier documentation as an example dataset as such we figured that it would be easily implemented and would offer good results to compare our decision tree classifier against the SKLearn DecisionTreeClassifier. The second dataset was the 'Secondary Mushroom' dataset [3] which contains hypothetical mushroom data with the target variable 'class' determining whether each mushroom was edible, poisonous, or undetermined. This dataset was chosen due to the large number instances and features to which could be compared to the Iris dataset. We tried to implement a third dataset, the 'Parkinsons Telemonitoring' dataset [4] which featured a range of biomedical voice measurements of 42 people with Parkinsons disease. The issue came from the fact that the target variables ('motor_UPDRS' and 'Total_UPDRS') were continous and not discrete which made it difficult to use with DT-One.

Below shows the number of instances for the three datasets and the number of features for each dataset:

Name	No. of Instances	No. of Features
Iris	150	5
Secondary Mushroom	61068	20
Parkinsons-Telemonitoring	5875	19

Methodology

DT-One is a custom Decision Tree classifier based off of the Classification and Regression Tree (CART) algorithm. At the core of the CART algorithm is a binary tree in with several different nodes. These nodes have differing functions. When given a new input from a dataset, the tree is traversed by the evaluation of the input at the 'root' node and then measuring the split from there. The CART algorithm relies on selecting input variables, the minimum split amount required to make a split from the nodes, and the maximum depth of the tree model. The splitting uses the idea of recursive binary splitting in which values are split at different values with a cost function resulting the best split being selected. The model also uses the Gini index to test the purity of the leaf nodes.

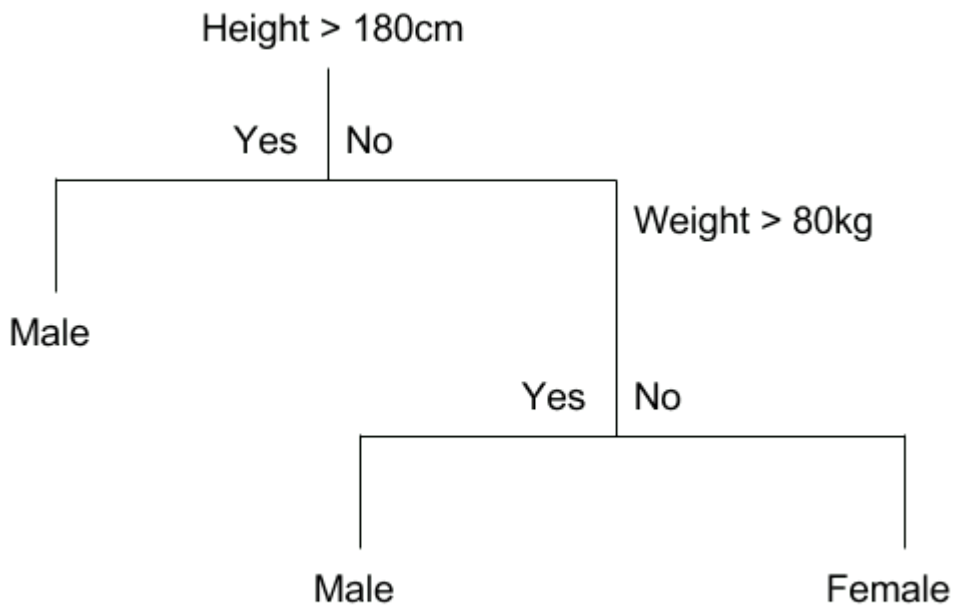

Figure: Example Decision Tree from Brownlee [5]
The basis of the DT-One is formulated around three classes; two of which are integral to the classifier and the third handles statistics , such as memory usage, time-tracking and saving performance metrics to a CSV file. The first class is the 'Node' class and it only represents a single node within the tree. It defines the index of the feature which is being split, and the threshold value for this split. It also defines the left and right sub-tree nodes to which the splits will go to. The value of the class label is also stored within this class.

The second class is the 'DTClassifier' class which is the implementation of the CART algorithm. This class features four constructors in its initialization (min_Samples_Split, max_Depth, random_state, root) The constructor min_Samples_Split defines the minimum number of required samples to split a node and max_Depth states the maximum possible depth of the decision tree. Random_State denotes the random integer which controls randomness and is designed to allow data to be reproducible. The root constructor defines the root node position for the tree. For this initialization function, we provided some default values which were:

```
Min Samples Split = 2
Max Depth = 2
Random State = None
```

The function 'mostCommonLabel' takes an input of Y which is an array and finds the most common label in a set of target labels. It also converts the array into a set which is designed to remove duplicate labels and allows the classifier to choose the most common label this is utilized through the max function. The 'fit' function is resposible for the actual training of the decision tree on the training data (Xtrain and Ytrain). First it checks the shape of the array Y and if it is still an array it converts it into a 2D column vector. Once reshaped it then concatentates the features and target labels into one dataset which then is passed to the function 'buildTree'. The 'buildTree' function is a recursive method which builds out the decision tree from the root node and continues to build until it fulfils its stopping criteria. The stopping criteria states that if there are fewer samples than the specified min_Samples_Split or the depth of the tree exceeds the specified max_Depth then the node must become a leaf with the most common label. The 'getBestSplit' function finds the best theoretical split based on the feature and the threshold which maximises the gain of information. This function loops through every feature in the feature index and applies all of the potential threshold values. The data is then split into one of two groups; the first group is data that is less than or equal to the threshold and

the second group is for data which is greater than the threshold. The information gain is calcualted in the 'informationGain' function which states that the information gain is the reduction of Gini impurity. The formula for this is seen below:

$$\text{Information Gain} = \text{Gini(parent)} - \left(\text{weight}_l \cdot \text{Gini(left)} + \text{weight}_r \cdot \text{Gini(right)}\right)$$

Where:

Gini(parent) = the Gini Impurity of the parent node.

weight_l & weight_r = proportions of the left and right child nodes with respect to the parent.

Gini(left) & Gini(right) = the Gini Impurities of the left and right child nodes.

The 'giniIndex' function calculates the Gini impurity which is utilised within the 'informationGain' function. It takes each class label in the target dataset (y) and computes the probability of the class and sums the total squared probabilites. It then applies the Gini index. The Gini index formula is:

$$\text{Gini}(S) = 1 - \sum_{i=1}^{k} p_i^2$$

Where:

p_i is the proportion of class i in the set S

k is the number of classes

The functions 'makePrediction' and 'predict' makes a prediction on a single input sample from the decision tree and then applies that prediction to a set of samples. The prediction making function is recursive and works from the root of the tree to the leaf node(s) and then returns the class label which is predicted for the sample. The final function of DTClassifer class is 'printTree' which is responsible for printing an actual representation of the decision tree. The tree is represented hierarchically showing each split and it also prints the feature index and threshold for the current node which is being split if it not a leaf node.

The third and final class of the decision tree is the 'timeMemory' class which is utilised to track time and memory usage during the decision tree process. The functions 'memoryUsage' and 'trackTimeMemory' returns the current memory usage o and tracks the current time taken during the fitting and prediction stages of the decision tree model. The 'saveToCSV' function is used to save the performance metrics of the model(s) to a CSV and the 'getSysInfo' function returns information about the system running the model. This function includes the operating system, CPU architecture, memory usage, and processor count. This was used to be able to quanitifably distinguish different systems which run the models in the aim of being abe to make easier and better conclusions.

Results

The following section shows the results on the Iris and Secondary Mushroom data from both DT-One and DT-Two (DT-One is represented as "[dataset] Data', DT-Two is represented as '[dataset] Data SK). The Parkinsons Telemonitoring dataset results have been omitted as mentioned in the Introduction. Whilst running both of the decision trees, we used a nested for loop to cycle through a set of specified minimum sample split and max depth values to see whether or not an increase in either of these values would positively impact the models performance. The following pages show a range of graphical representations of the data obtained from both models.

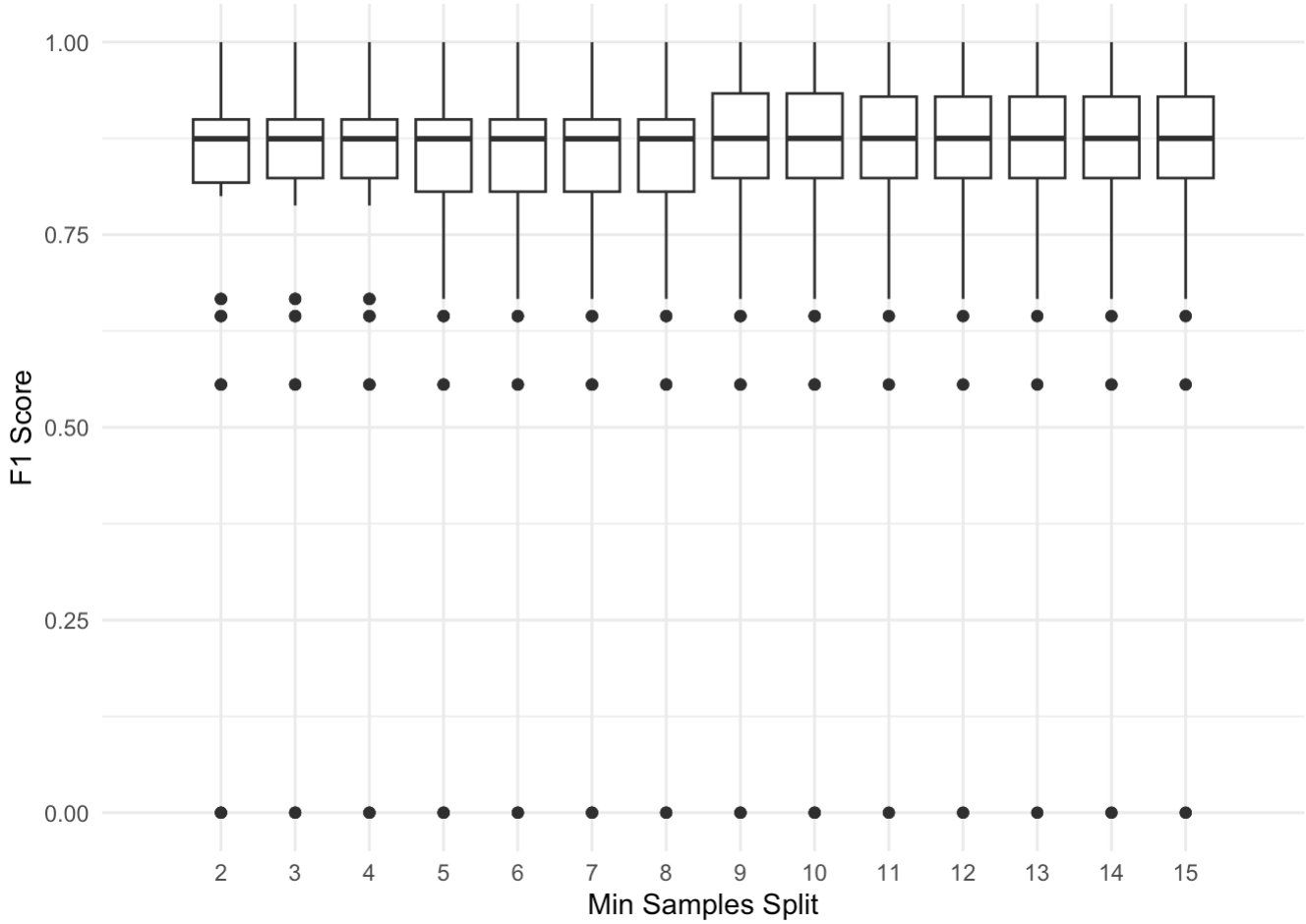

Figure: Boxplos of Min Samples Split against F1 Score

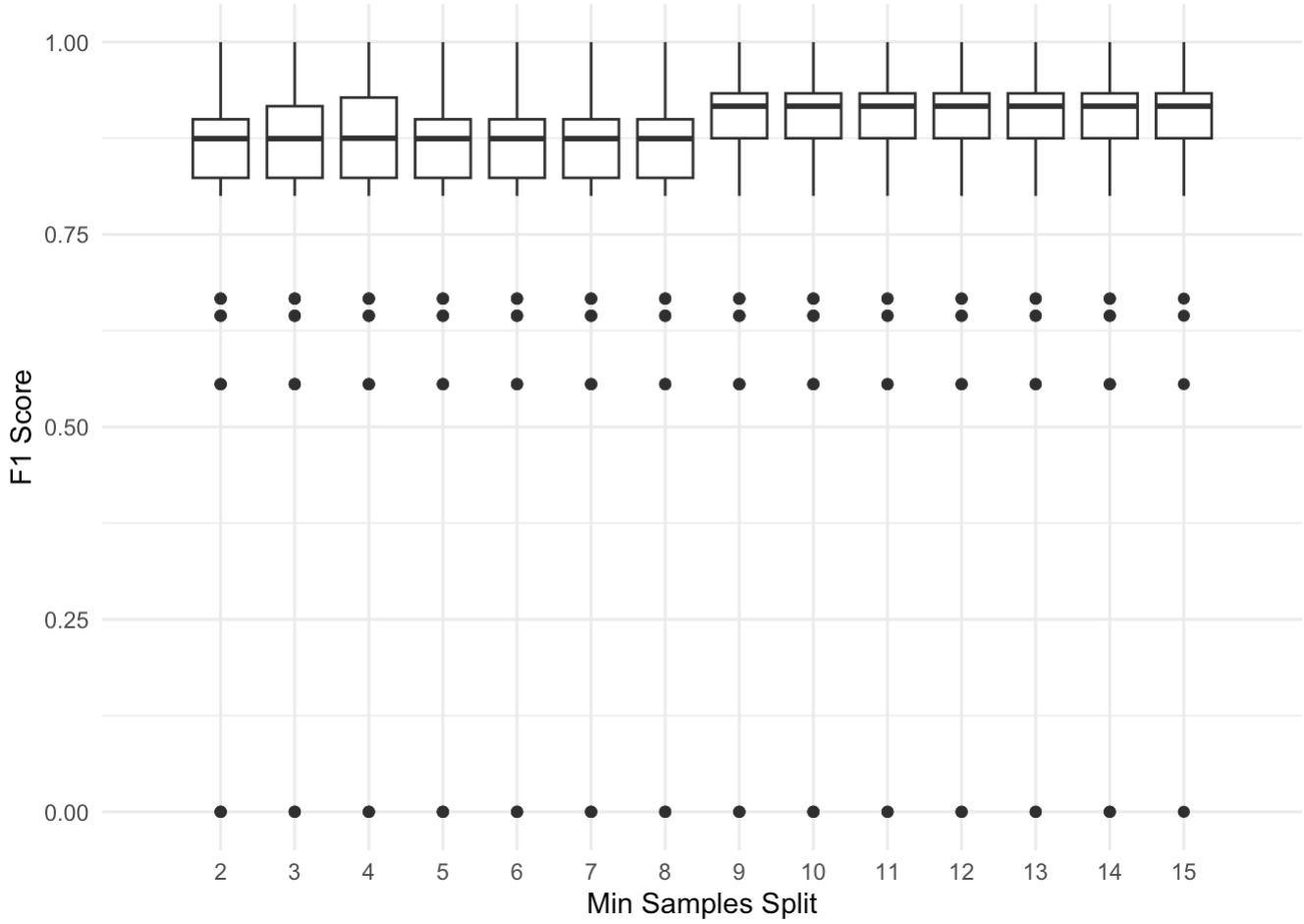

Figure: Boxplos of Min Samples Split against F1 Score

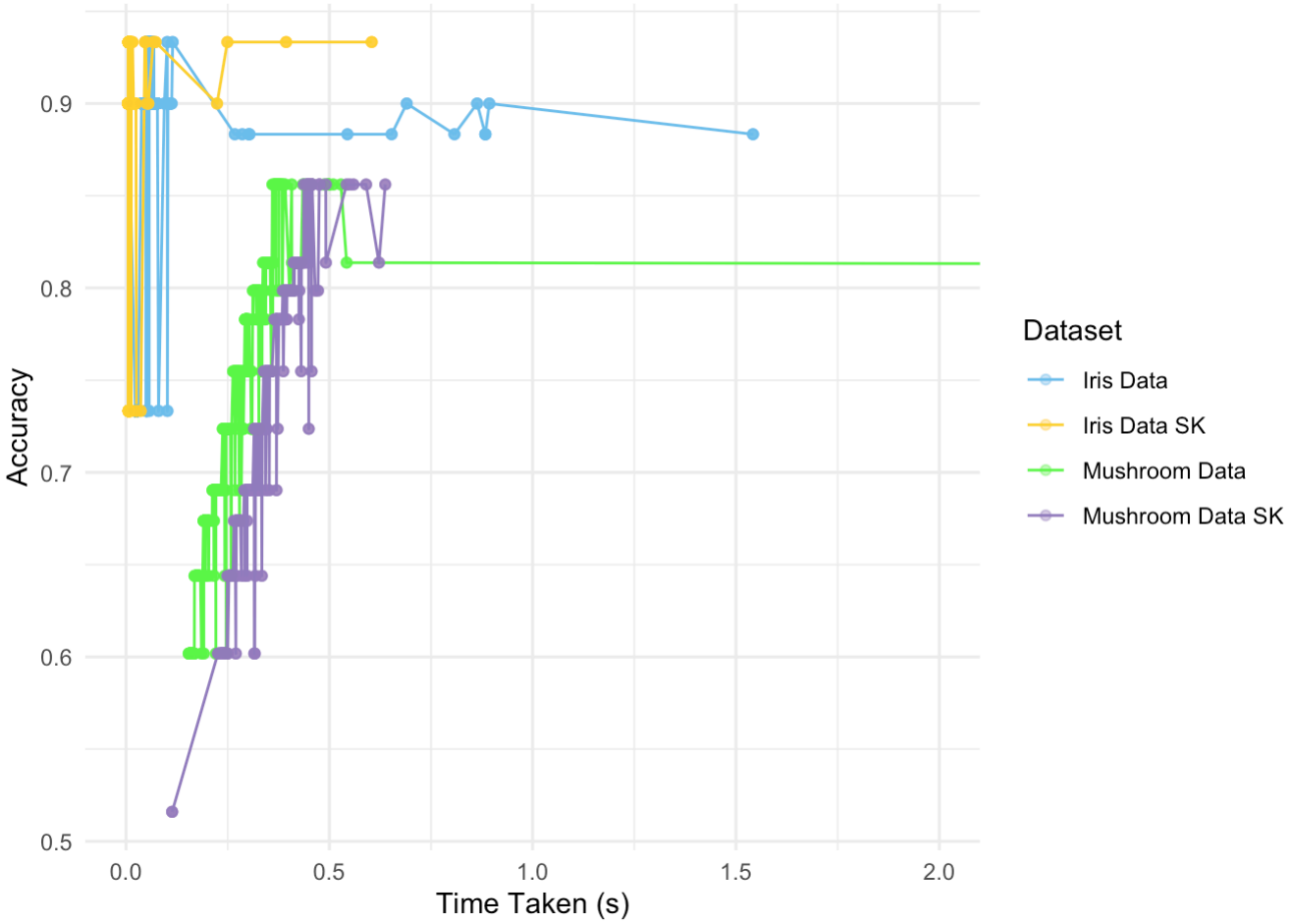

Figure: Line Graph of Time Taken versus Model Accuracy

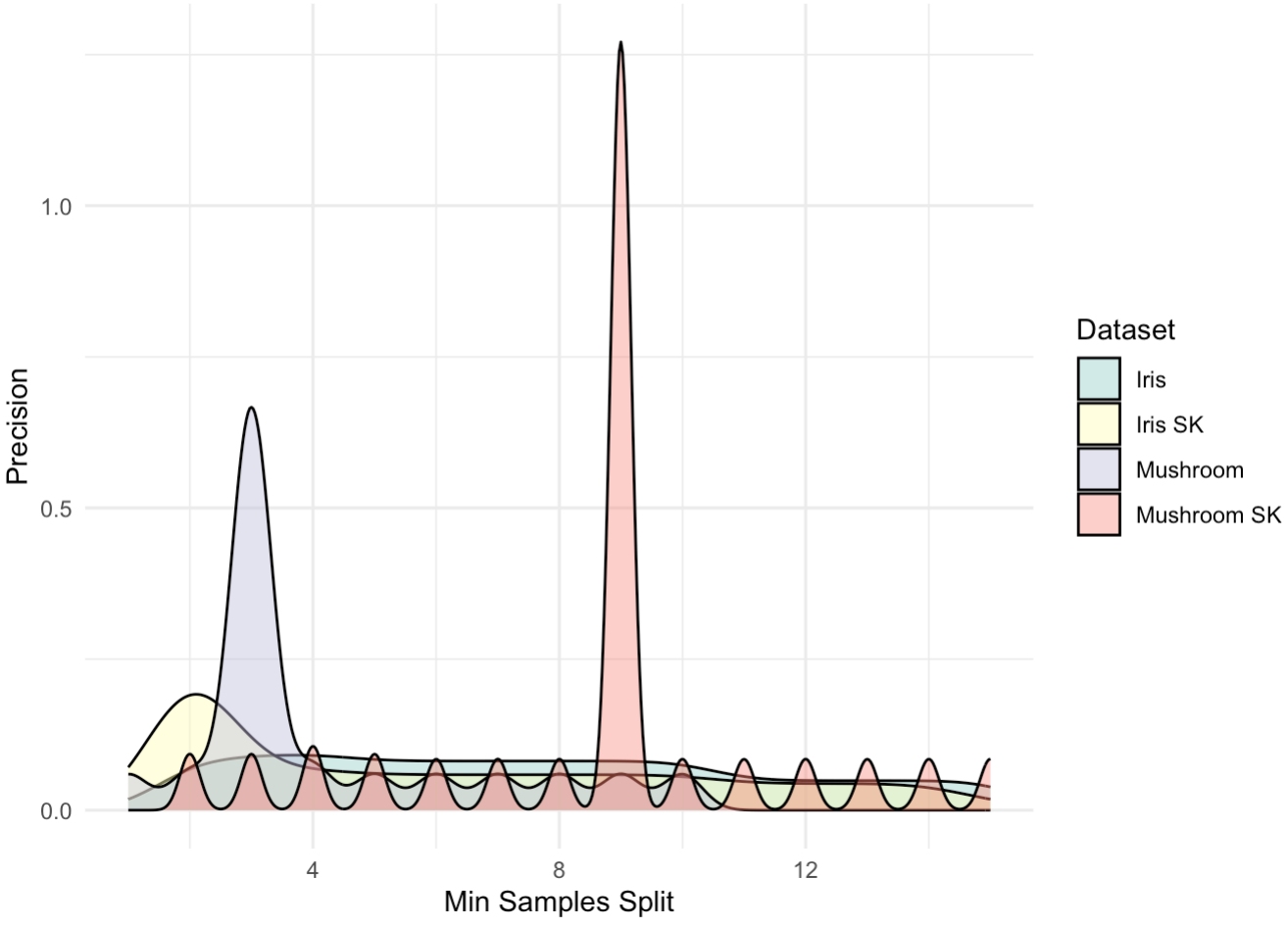

Figure : Min Samples Split against Precision

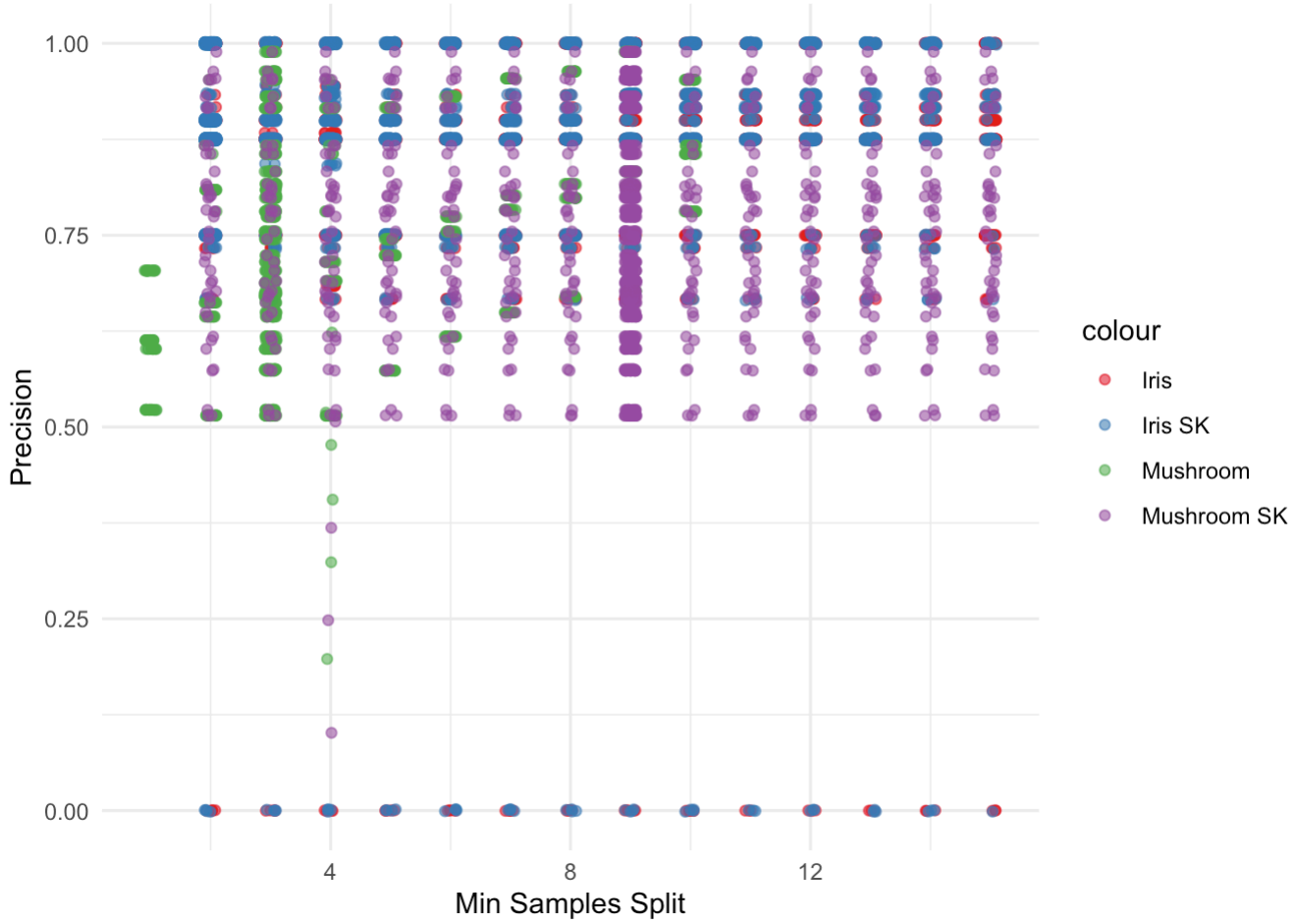

Figure: Scatterplot of Min Samples Split against Precision

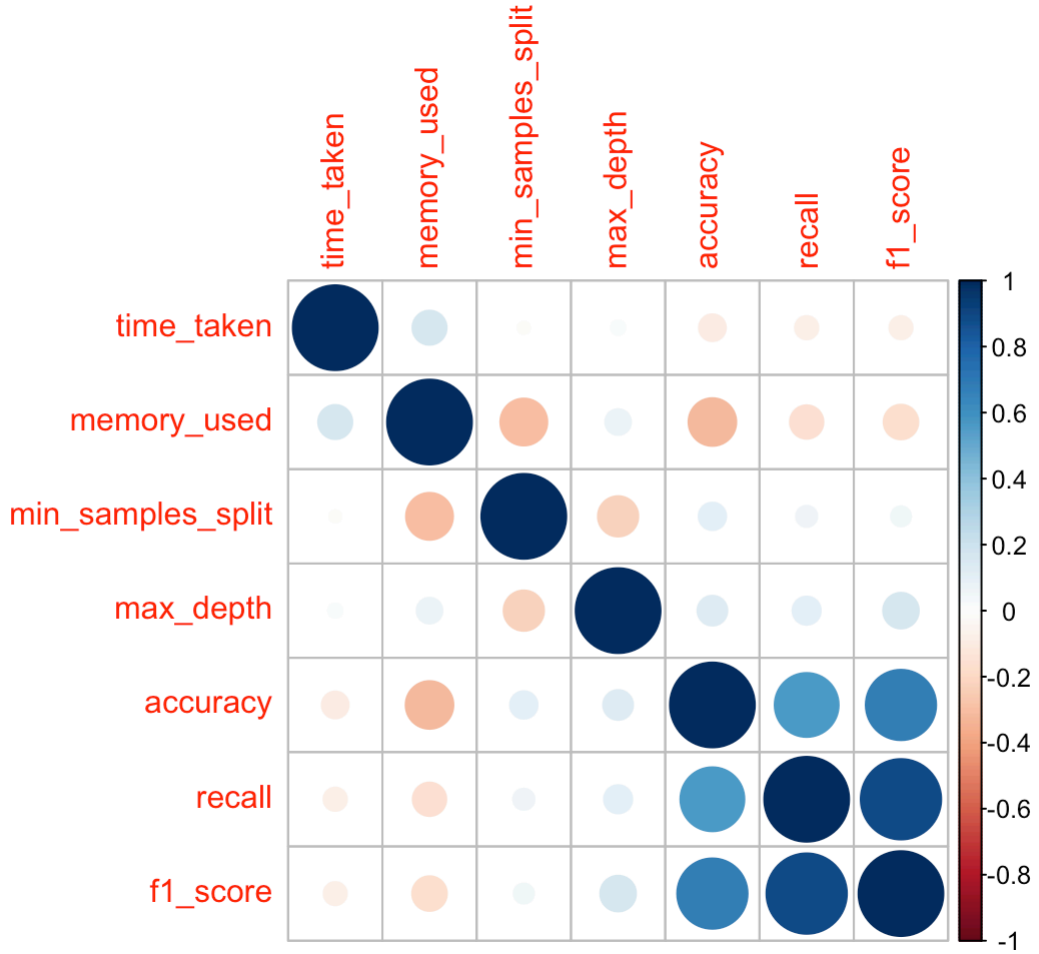

Figure: Correlation Matrix

```
## $anovaAcc
##               Df Sum Sq Mean Sq F value Pr(>F)
## dataset        3  41.42  13.806    3792 <2e-16 ***
## Residuals   8437  30.72   0.004
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaF1
##               Df Sum Sq Mean Sq F value Pr(>F)
## dataset        3  27.59   9.195   525.4 <2e-16 ***
## Residuals   8437 147.67   0.018
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaIris
##                     Df Sum Sq Mean Sq F value Pr(>F)
## min_samples_split    1   0.03 0.03371   1.462  0.227
## Residuals         3073  70.84 0.02305
##
## $anovaIrisAcc
##                     Df Sum Sq  Mean Sq F value Pr(>F)
## min_samples_split    1  0.031 0.030647   11.26  8e-04 ***
## Residuals         3073  8.361 0.002721
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaIrisSK
##                     Df Sum Sq Mean Sq F value Pr(>F)
## min_samples_split    1   0.01 0.01335   0.672  0.413
## Residuals         3068  60.99 0.01988
##
## $anovaIrisSKAcc
##                     Df Sum Sq Mean Sq F value   Pr(>F)
## min_samples_split    1  0.061 0.06084   24.16 9.35e-07 ***
## Residuals         3068  7.727 0.00252
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaMushroom
##                     Df Sum Sq Mean Sq F value Pr(>F)
## min_samples_split    1  2.276  2.2760   465.5 <2e-16 ***
## Residuals         1152  5.632  0.0049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaMushroomAcc
##                     Df Sum Sq Mean Sq F value Pr(>F)
## min_samples_split    1  2.248  2.2477     507 <2e-16 ***
## Residuals         1152  5.107  0.0044
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## $anovaMushroomSK
##                     Df Sum Sq  Mean Sq F value Pr(>F)
## min_samples_split    1  0.000 0.000150   0.022  0.883
```

```
## Residuals         1140  7.884 0.006916
##
## $anovaMushroomSKAcc
##                     Df Sum Sq  Mean Sq F value Pr(>F)
## min_samples_split    1  0.002 0.001627   0.258  0.611
## Residuals         1140  7.185 0.006302
```

Discussion

The ANOVA tests shown above show that there is an extremely small p-value (<2e-16) in the data overall which suggests that the differences between the different datasets are statistically significant which in turn suggests that the different variations of decision tree classifier directly affect the accuracy and F1 score of the overall data. However, within the Iris SKLearn dataset there is evidence of a significant effect from the minimum sample split value on the F1 score which differs from the Iris dataset where the p-value between min_samples_split and F1 score is 0.227. This is also seen within the Mushroom and Mushroom SKLearn datasets where in the Mushroom dataset's p-value is 0.413 which suggests no significant effect in comparison to p-value of <2e-16 for the Mushroom SKLearn dataset. This p-value is extremely low and therefore indicates that there is a very significant effect on the F1 score. There is also an extremely strong signficance between the minimum sample split and accuracy in this dataset which suggests that this dataset is very sensitive to differing chances in the minimum sample split value.

The correlation matrix represents an aggregation of all of the data into a single dataset. There is a somewhat neagtive correlation between the minimum sample split size and the memory usage, which suggests that as the split increases increases the memory usage appears to decrease. There is a low correlation between the minimum samples split and accuracy which in turn shows that a greater sample split minimum does not strongly affect model accuracy. There is a slight negative correlation between max depth and minimum sample split overall, which shows that as the maximum depth of the tree increases it the minumum sample split becomes less important and this shows an interplay between these two variables. There is a low negative correaltion between accuracy and time taken which shows that the actual training time does not improve accuracy. This can also be said for model recall and F1 score.

There is evidence of some positive correlation between maximum depth and accuracy, recall, and F1 score. However this correlation is only small but does suggest that a deeper depth could capture more complexity which in turn could lead to an improvement in model performance. There is a strong positive correlation between model recall and model accuracy which shows that as the model makes accurate predicition it is also having a higher recall. This correlation is also shared with the F1 score which in turn suggests a good balance between model precision and model recall.

Conclusion

In conclusion this report has shown analysis of dataset obtained from two separate decision tree classifier algorithms. The DecisionTreeClassifier from the SKLearn package is conclusively better than the custom decision tree classifier that we implemented but it does lack some of the features (such as performance metrics, time and memory tracking) which we had to implemenent from our own decision tree classifier to make the results comparable. Whilst creating and tuning hyper-parameters, we believed that there could be a strongly correlation between the minimum sample split size and the model accuracy this however was disproved and our analysis in fact showed there is a greater importance on the depth of the decision tree itself. The positive correlation between recall, accuracy and F1 score also demonstrated that a higher model recall will positively impact the accuracy and overall F1 score of the model.

Bibliography

[1] R. A. Fisher, “Iris.” archive.ics.uci.edu, 1988. Available: https://archive.ics.uci.edu/dataset/53/iris (https://archive.ics.uci.edu/dataset/53/iris)

[2] R. A. FISHER, “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS,” *Annals of Eugenics*, vol. 7, pp. 179–188, 1936, doi: 10.1111/j.1469-1809.1936.tb02137.x (https://doi.org/10.1111/j.1469-1809.1936.tb02137.x).

[3] D. Wagner, D. Heider, and G. Hattab, “Secondary mushroom.” archive.ics.uci.edu, 2023. Available: https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset (https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset)

[4] A. Tansas, M. A. Little Max, and L. Ramig, “Parkinsons telemonitoring.” archive.ics.uci.edu, 2009. Available: https://archive.ics.uci.edu/dataset/189/parkinsons+telemonitoring (https://archive.ics.uci.edu/dataset/189/parkinsons+telemonitoring)

[5] J. Brownlee, “Classification and regression trees for machine learning.” Machine Learning Mastery, Sep. 2017. Available: https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/ (https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/)