SCC461FinalReport

36506921

2024-01-14

Abstract

In this project I have made a decision tree classifier based on the mathematical principles of the `C4.5` model. And compared this classifier with the decision tree classifier in sklearn library.The overall result shows that my classifier is much less efficient than sklearn but only a little bit worse than it in terms of performance case.The decision tree classifier in sklearn is based on `CART` which is slightly different from the `C4.5` model. I compared the operational efficiency and performance of my model with sklearn's model by running iterative simulations on different datasets with different hyperparameters and other variables, and estimating the influence of some parameters. Through linear regression analysis, I found that the number of features in the dataset and the number of instances significantly affected the model's run efficiency, and that these two features together explained about 17% of the variability in run time. In addition to this, the analysis of memory usage, and performance case parameters such as accuracy, showed that my model leaves much to be desired. These findings reveal potential reasons for the performance variance and provide illuminating perspectives for further optimisation of the classifier.

Introduction

With the increasing amount of data, the use of machine learning algorithms, especially classifiers, is becoming more and more widespread. Decision trees, a popular classification method, are popular for their concise logic and easy-to-explain decision-making process. However, as the dimensionality of the data increases, the decision tree construction and prediction runtime can rise dramatically, especially in resource-limited environments. Performance optimisation becomes central to enabling fast and accurate data analysis. Therefore, understanding the core theory of different decision tree classifier algorithms and putting it into practice, comparing the performance differences between different algorithms, can provide practical guidance for algorithm selection and parameter tuning.
Sklearn is one of the most famous machine learning libraries in Python, and its classifiers are widely used because of their stability and efficiency, so in this study I choose the decision tree classifier in sklearn as a comparison sample with the one I made myself. The goal of this study is to compare the performance of my decision tree and sklearn classifiers on different datasets through a series of theories and experiments, and to analyse their impact on the running time. Theoretically, the `C4.5` classifier performs node creation for decision trees based on the ratio of pairwise information entropy and information gain, whereas sklearn's decision tree is based on the `CART` algorithm, which performs node creation and differentiation based on Gini Index. Based on this theoretical distinction, I analysed the possible differences. Then, I conducted an exhaustive comparison test between the two classifiers, recording performance data under different configurations of number of features and number of instances, and applying statistical models for in-depth analysis. By constructing linear regression models, I not only quantified the impact of each parameter, but also tested its statistical significance.
I found that theoretical differences lead to the fact that these two decision tree classifiers are adapted to different kinds of data processing tasks respectively, and therefore will differ when analysed with the same dataset. Also, the inconsistency in the complexity of the computation of the Gini coefficients and the computation of the information gain ratios is taken into account, which also leads to a difference in efficiency. This is also largely consistent with the results of the actual experiments. Although the efficiency of my model is much lower compared to the efficiency of sklearn built-in function due to the limitation of programming

ability, the comparison of the difference in accuracy also reveals that the `C4.5` model is also excellent in performance. If I can optimise my model, such as increasing the memory footprint to reduce the time cost, and reducing the overall time and space complexity of the model to make it optimal, then it will definitely be able to reach or even exceed the efficiency and performance of sklearn's internal `CART` algorithm.

Methodology

In this section I will explain my decision tree classifier algorithm in detail. My decision tree classifier is based on the `C4.5` algorithm, but it is not exactly the same. The full `C4.5` decision tree classifier is too complex, and I have only implemented its basic classification method in this article, and have not implemented complex features such as pruning and generating classification rules. Decision tree based classification algorithms all have a roughly the same logic, which is to select the optimal classification attribute for classification, and create a decision tree through repeated iterations that can be used to predict the classification of new instances. Common decision tree algorithms are `CART`, `C3` and `C4.5`. Among them, `CART` is based on Gini coefficient for classification, while `C3` and `C4.5` are based on information entropy correlation computation. $C4.5$ is an upgraded version of `C3`, which was proposed by Ross Quinlan in 1993, and it can solve some defects of `C3`. Therefore in this project I chose to make `C4.5` decision tree classifier and compare it with sklearn built-in decision tree classifier based on `CART`, hoping to analyse what are the differences between the algorithms that classify based on different metrics. My algorithm includes the following steps:

1. data preprocessing
2. select the optimal segmentation features based on the information gain ratio
3. Recursively constructing a decision tree
4. Prediction using the decision tree

Data Pre-processing

For the different characteristics of different datasets, I added the handling of missing values in the `load_dataset` function of the main file (`main_run.py`), which fills the missing values after determining that there are missing values. The logic for the filling is to fill the missing values according to the plurality of the column of features in which the missing values are located. After filling, the data is partitioned and divided into training and test sets according to a predetermined ratio. Moreover, the basic types of data are classified as `Integer`,`Continuous` and `Categorical`. I use the first two types as numerical features and other types as categorical features. In the subsequent algorithm, the classification of subsets is done according to two different logics.

Selection of optimal features

In my algorithm, I choose to make the selection of optimal segmentation features based on the information gain ratio. First of all, I need to list the concepts and calculation formulas related to information gain ratio: Information entropy describes the degree of information chaos in the data set, the lower the information entropy, the more ordered the samples are, and the higher the purity of the samples. The calculation formula of information entropy is:

$$E(D) = -\sum p_i log_2 p_i$$

where p_i is the relative frequency of the ith class in the dataset. Information gain refers to the change in information entropy brought about before and after the division of the dataset according to a certain feature value. That is, the change of information entropy of the output columns of the dataset D after the feature A is known. The formula for information gain is:

$$Gain(D, A) = E(D) - \sum \frac{D_v}{D} E(D_v)$$

The information gain ratio is a joint calculation of the ratio of the information gain calculated from the known feature A and the information entropy of the feature A itself. The formula is as follows:

$$Gain\_ratio(D, A) = \frac{Gain(D, A)}{Split\_info(A)}$$

Where $Split\_info(A)$ is the information entropy of feature A column. Based on the above formula, the `calculate_information_entropy` method in the `C45` class defined in my algorithm is used to calculate the information entropy. And the `calculate_information_entropy` method in the `find_best_feature` method is used to perform the calculation of the information gain ratio by calling the `calculate_information_entropy` method. The feature with the maximum information gain ratio is found by cycling through the calculation of the information gain ratio for each feature, and after finding the best feature, I eliminate this feature from the list of feature values and continue to the next iteration. While this is in a sense lowering the upper limit of the model's performance, it is also a limit on the depth of the tree. When calculating the information gain and the information entropy of the features themselves, for features defined as numeric in the data preprocessing stage, I use the `calculate_split_continue` method to find the optimal split ratio by cycling through the minimum post-split weighted variance, using quintiles or other quintiles as the split points.

Recursive Building Decision Trees

In my model, I call the `build_tree` method by using the method `fit` on the training set to keep creating child nodes in a recursive logic and set the conditions for judging as a leaf node:

1. minimum information gain ratio: when the information gain ratio after segmentation is less than this bound, the node is defined as a leaf node and iteration is stopped;
2. feature column empty: means that all features have been used and iteration is stopped on the logical premise of not reusing features;
3. maximum number of instances in the dataset: when it is less than the maximum number of instances, the node contains a small enough number of strengths to stop the iteration.

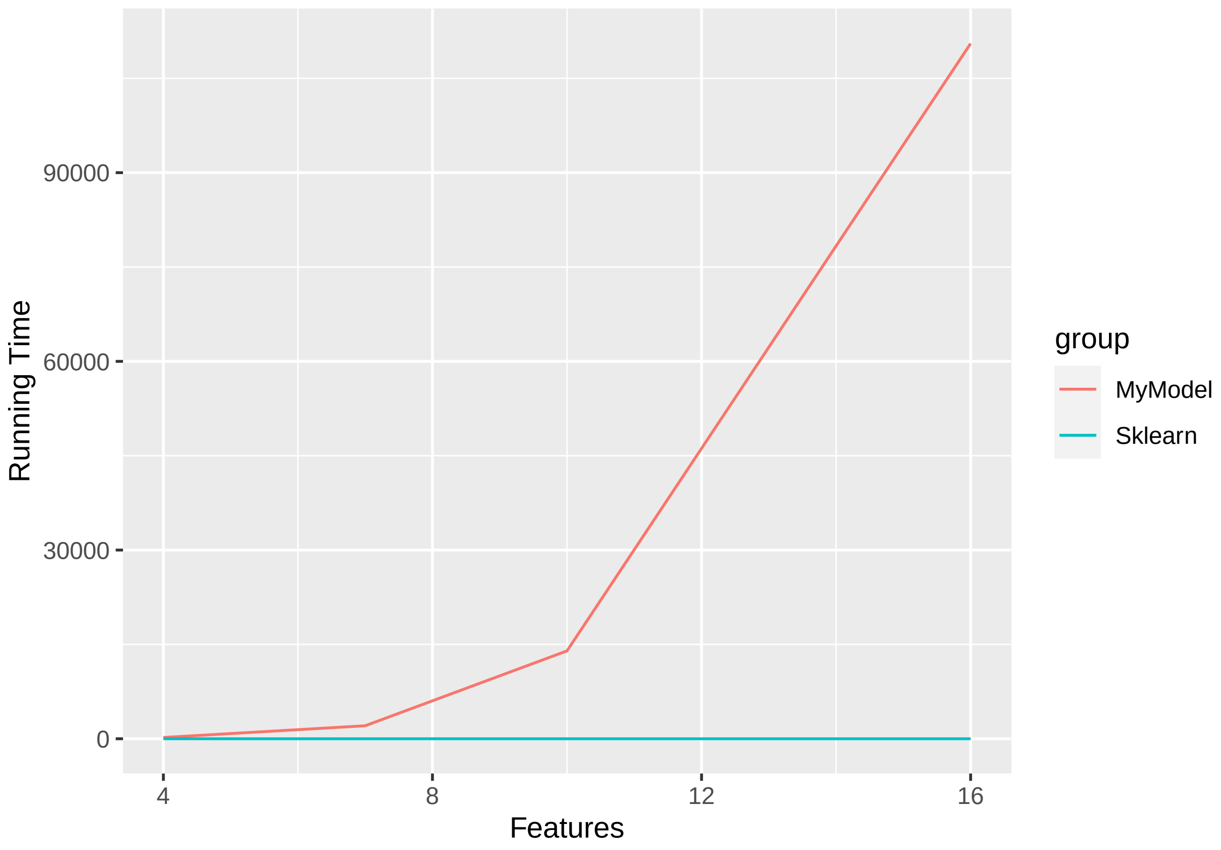
In addition, in the method of constructing the decision tree, I make a distinction between the types of features; when the feature is `Integer` or `Continuous`, the child nodes are divided into left and right nodes. The child nodes of features of other categories are then classified according to the unique value of the feature.

Prediction using decision trees

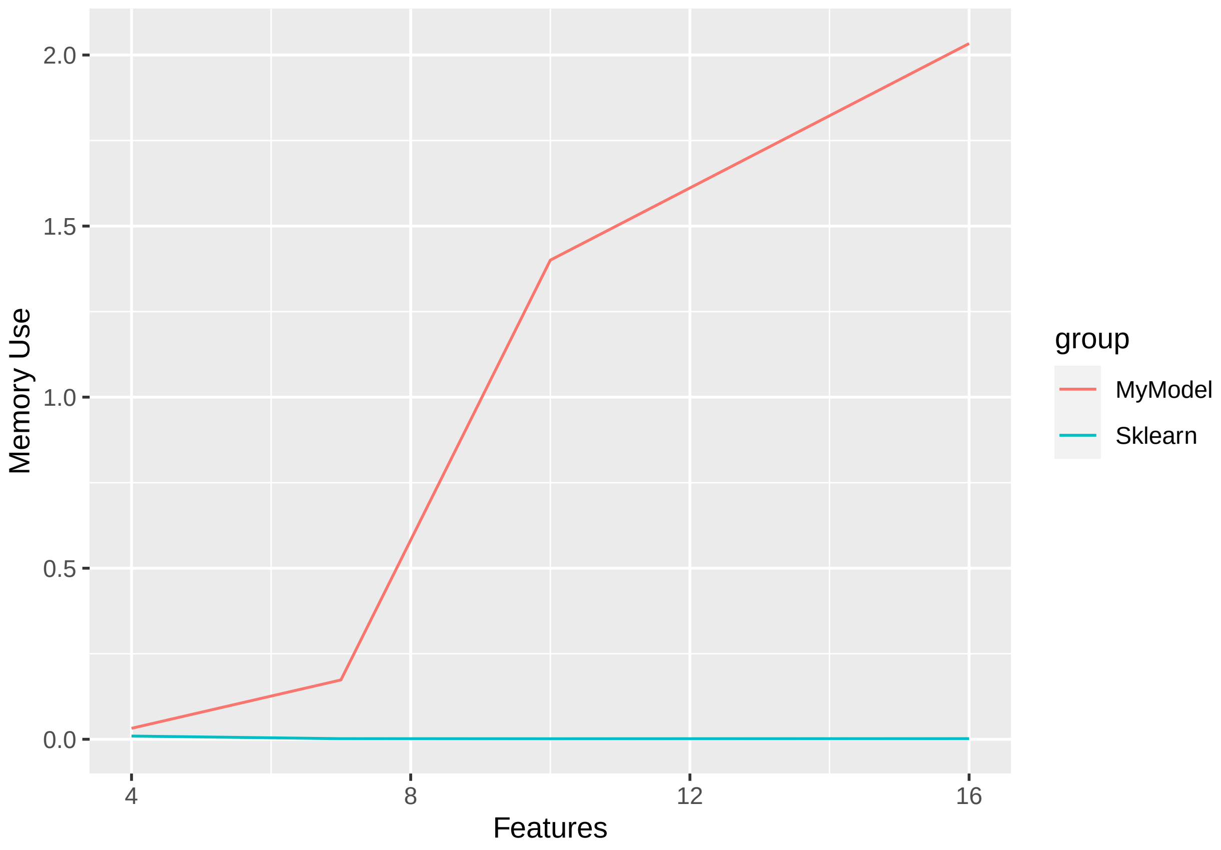
After the decision tree construction via the `fit` method, it is time to predict using the `predict` method. In this method, I ran the decision tree for each instance in the test set using the `apply` method in conjunction with an anonymous function and stored the results in `result`.

Results

When analysed from an overall perspective, my model differs significantly from the decision tree classifiers in the sklearn library in terms of time efficiency, with an average running time of $2.5768 \times 10^4 ms$ when targeting different scenarios, whereas the decision tree classifiers in the `sklearn` library run in $73\mu s$, a difference of more than eight orders of magnitude between the two. Clearly my model has a lot of room for upgrading. The time taken by my model to construct a decision tree shows a clear positive correlation with the number of features, and as the number of features increases, so does the growth rate in the time taken.

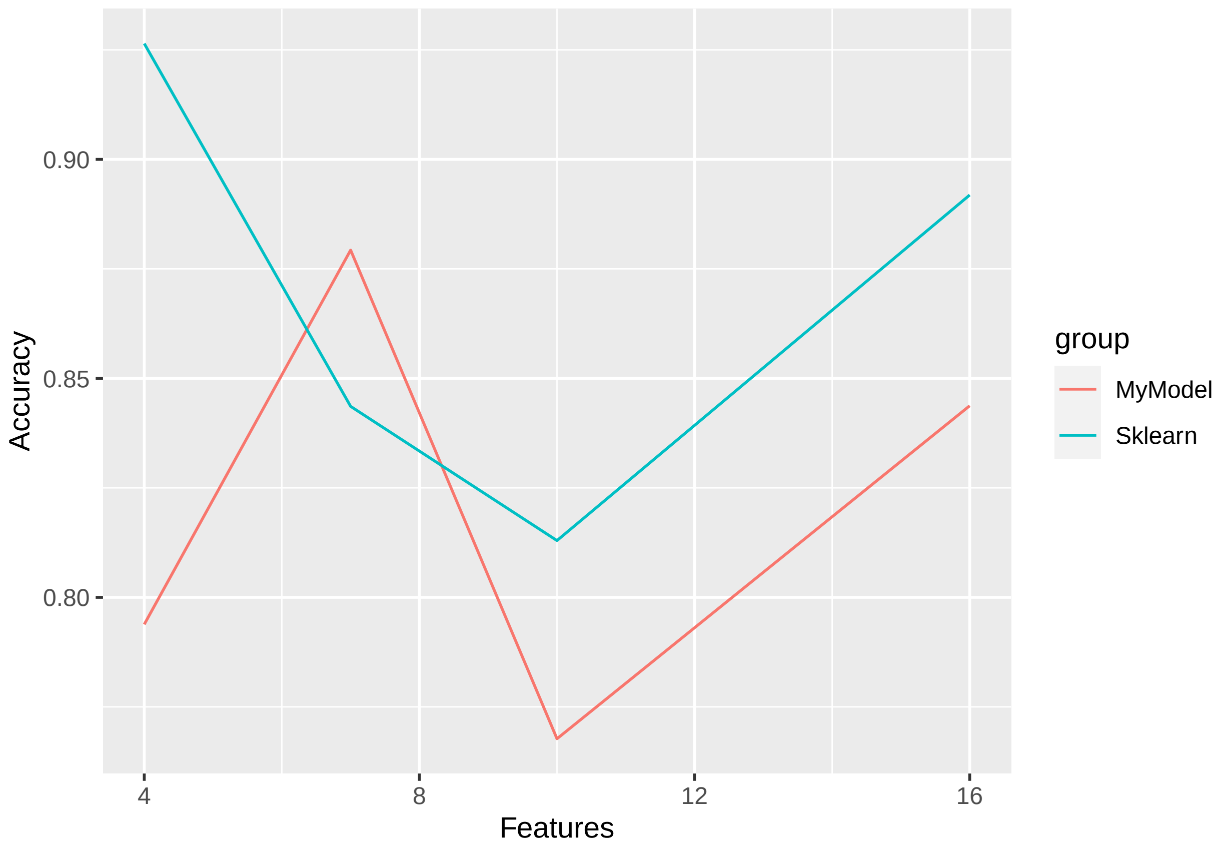


In terms of memory usage, my model has the same problem and sklearn remains far more efficient.



There are several reasons for the above discrepancy based on the analysis:

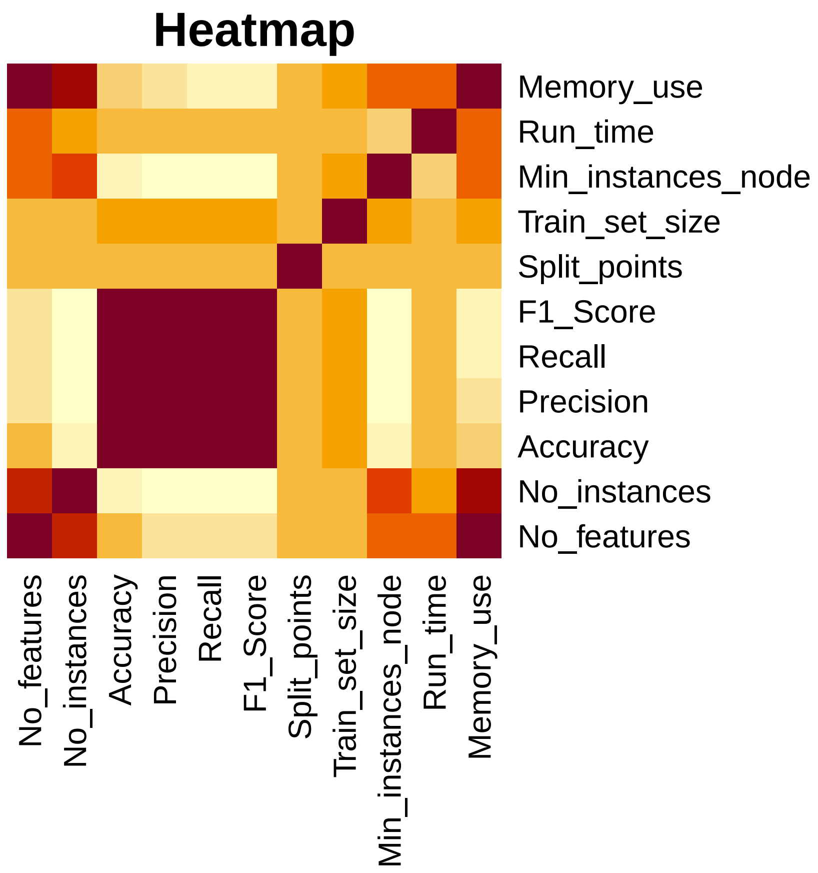
- I did not add a pruning method, which would have prevented me from removing redundant nodes;
- Only a few quantile points were considered when calculating the split points for numerical features, this simple split logic when analysing particular data with a large skewness in the distribution can lead to differences in accuracy;
- The computational complexity of the Information Gain Ratio is higher than the Gini Index, so even with the same spatio-temporal complexity of the code, my model is a bit slower.



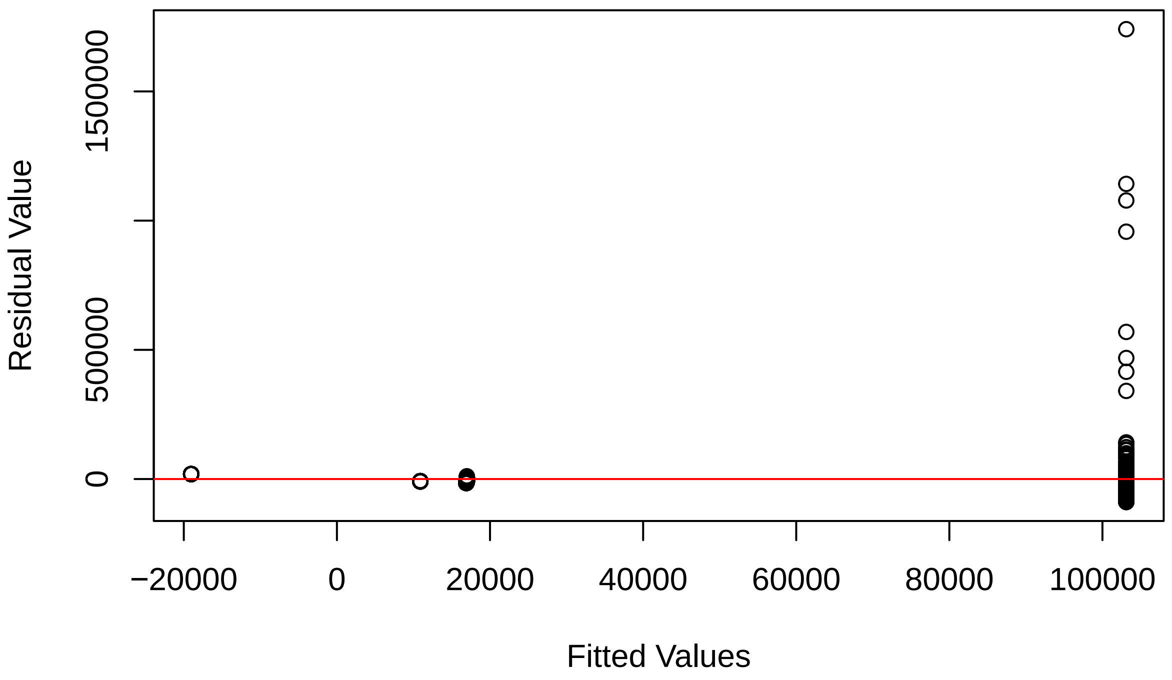
In addition, there are four features that can be analysed in terms of the performance of the model's prediction results, which are accuracy, precision, recall and F1 score. According to the formula of F1 score it can be considered that the correlation between precision and recall and F1 score is very high. Moreover, a simple division calculation reveals that accuracy is very similar to F1 score, and the mean of the two is divided as 1.0079773. Therefore, only the accuracy feature can be analysed. The graph below shows that in terms of model performance, the gap between my model and sklearn model is not that big, and even higher than sklearn model in some cases.

Discussion

In the previous subsection, I briefly described the results of the analysis. In all respects, my model does not perform as well as sklearn's decision tree classifier, which is of course to be expected. As a very commonly used library in the machine learning field, sklearn's built-in functions are obviously highly optimised. So it's not realistic for me to try to achieve such a level of optimisation based on my own personal ingenuity alone. However, this does not mean that I don't need to analyse the shortcomings of my model, the significance of the analysis is to improve my own programming level and deepen my theoretical understanding of the decision tree algorithm. Firstly, when I wrote the whole algorithm, I did not consider reproducing the complete `C4.5` algorithm, but simplified it, so the gap in efficiency and performance results may originate from this. I omitted the pruning part. Pruning is an approach that can improve the time and space efficiency of an algorithm. When the amount of data is too large, the branches of the tree may extend into some redundant directions, leading to a decrease in the overall efficiency of the algorithm, so many algorithms make use of pruning to make efficiency gains. Therefore, in my algorithm, the redundant branches caused by the lack of pruning leads to the overall efficiency reduction. And I have adopted a relatively simplified approach to many details, such as the split point calculation for continuous values. My algorithm uses quantile points, such as quintiles, to perform circular calculations to find the point with the lowest weighted variance among these quantile points as the best split point. However, this is not suitable for datasets with very large skewness. If it can be changed to bisection or other more efficient methods, my algorithm will be more efficient and the prediction results will be more accurate. Secondly, after analysing the experimental data in a heat map I found that not every change in hyperparameters or other variables resulted in a change in efficiency or effectiveness. As can be seen from the heat map, the features that had the greatest impact on the efficiency of the model I constructed were: the number of features, the number of samples, the segmentation of leaf nodes, and the proportion of the training set. The features that have the greatest impact on the performance of my model are the proportion of the training set and the calculation of the split points for continuous feature variables. The other features had very little impact, which suggests that there are some logic issues in my algorithm that are causing certain variables to not work as well as they should. If there is a subsequent opportunity to be able to continue optimising my algorithm, I would consider increasing the weighting of these variables as a way to improve the performance of the algorithm.



For time efficiency, I only study the effect of the number of features and the number of samples on time efficiency. I analysed the time efficiency using linear regression to see the effect of different features on time efficiency. The graph of the residuals from the analysis shows that most of the data points seem to be randomly distributed around the horizontal line ($y = 0$), which suggests that my model's predictions for these data are relatively accurate. This is because good residual plots have residual points that are completely randomly distributed. There are a few obvious outliers, especially where the fitted values are large. The residual values at these points are very high, indicating that the model's predictions at these points differ significantly from the actual values. This suggests that the model may not be capturing some non-linear relationships in the data, resulting in residuals that are not perfectly randomly distributed. Alternatively, there may be outliers or outliers in the data, and these values may have had a disproportionate impact on the regression results.

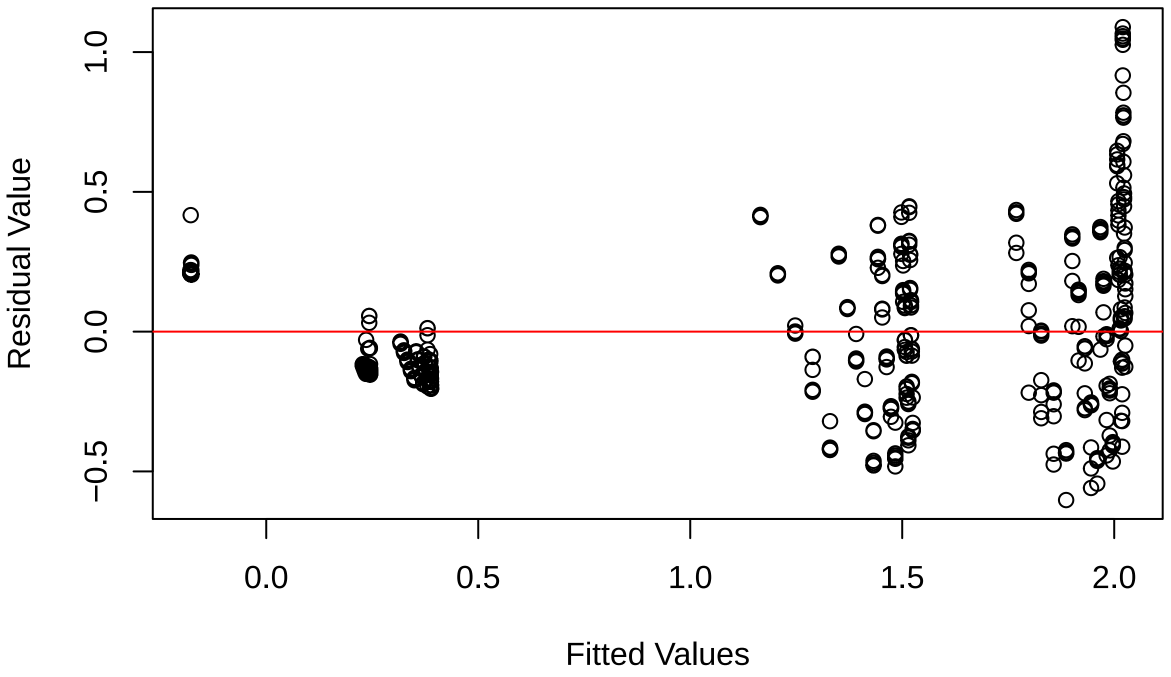


The linear regression model is summarised below:

```
##
## Call:
## lm(formula = Run_time ~ No_features + No_instances, data = my_model_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -89107  -14745   -8564   19185 1740845
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -6.871e+04  7.264e+03  -9.459  < 2e-16 ***
## No_features   1.250e+04  1.034e+03  12.080  < 2e-16 ***
## No_instances -2.065e+00  5.612e-01  -3.679 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 90760 on 997 degrees of freedom
## Multiple R-squared:  0.1693, Adjusted R-squared:  0.1676
## F-statistic: 101.6 on 2 and 997 DF,  p-value: < 2.2e-16
```

As can be seen from Coefficients, the intercept term is approximately equal to 0, and the running time increases by an average of 12500 units of time for each additional feature, and decreases by an average of 2.065 units of time for each additional instance. This is somewhat contrary to common sense, as increasing the number of instances increases the running time, and in combination with the curves for the number of features and time, it is reasonable to assume that the large number of instances with a small number of features in the dataset is influencing the conclusions. Therefore to draw more accurate conclusions, more precise control variables for the experiment are needed. The standard deviation of the number of features fitted is very large, up to 1034, which proves that the estimated coefficients are imprecise. The t-value is used to test whether each coefficient is significantly different from zero. Here the t-value for the number of features is 12.08 and the t-value for the number of instances is -3.679. These large absolute amounts of t-values indicate that the corresponding coefficients are very likely to be significantly different from zero. The

R^2 is 0.1693, meaning that the two independent variables explain about 16.93% of the Run_time variability. It can be concluded that the model does not fit the data very well and needs to be optimised.The F-statistic is 101.6 and the corresponding p-value is very small ($< 2.2e - 16$), which suggests that at least one of the predictor variables of the model is significantly influencing the response variable.



For the analysis of memory usage I also use linear regression analysis. Based on the heat map, I analysed the segmentation of leaf nodes, the number of features and the number of samples. The result of the analysis is shown in the figure above, which is a residual plot with a very poor distribution, the points in the plot do not show a random distribution, which means that the memory occupancy may need more variables to explain, or there is a non-linear relationship between these three variables and the memory occupancy. Therefore, it is necessary to increase the weight of each variable by replacing it with another regression model to simulate the curve, or by optimising the existing model to achieve a better prediction.

Conclusion

In summary, this report presents a comparative analysis between a decision tree classifier written inspired by the C4.5 model and the well-established sklearn decision tree classifier. The empirical findings show that although my model lags behind sklearn in terms of efficiency, the performance gap is not that large. And through linear regression analysis, it was found that the number of features and the number of instances had an impact on the runtime efficiency, which accounted for about 17% of the change in runtime. Despite the lower efficiency due to programming capacity limitations, this study provides some insight into the performance differences of the classifiers and identifies directions for further optimisation. Future work may focus on refining the model by pruning, considering more complex attribute segmentation strategies, and tuning hyperparameters to improve the potential performance of the model.

Bibliography

Analysis of variance
What isprun?