SCC.461 Final Assignment

36065118

24/01/2022

1 Abstract

Decision Tree is a data mining based architecture that is widely used for solving classification and regression problems. It classifies the data set by dividing the data space into sub-data spaces while forming an inverted tree-like structure comprising of leaf nodes, internal nodes and a root node. Decision trees are simple, intuitive and easy to implement models. The decision boundaries formed by the decision trees are parallel to the features if used as axes. They are also believed to handle large and complicated data sets efficiently.
This report discusses various algorithms to implement a decision tree along with a novel implementation of a decision tree classifier which is then compared with a library implementation of the same by scikit-learn, based on various performance metrics. The approach and the methods utilized are also discussed in detail.
keywords: decision tree, entropy, gini, data mining, classification

2 Introduction

All the points in a data set are represented using an identical set of features. These features can be continuous, categorical or binary in nature. If the corresponding correct outputs are provided with the data points then the learning becomes supervised, in contrast to unsupervised learning where the data points are unlabeled (Jain, Murty, and Flynn 1999).

Classification is usually a supervised learning technique that aims to assign a class label to an input data sample. The class label represents one of the given set of classes (Murty and Devi 2015). Classification techniques are classified as generative, which calculate the joint probability of the inputs and the labels, and discriminative, which model the posterior probability directly (Ng and Jordan 2002). Examples of discriminative classifiers include SVM, KNN, Logistic regression and Decision Trees. Examples of generative include Naive Bayes, Hidden Markov Models, LDA and Neural Networks.

First introduced in the 1960's, the Decision Trees are considered to be one of the most efficient algorithms for data mining applications. They are now utilized in a plethora of different applications (Friedman et al. 2001) as they are easy to implement and understand, they are robust even if the data in consideration contain some missing values. Decision trees also have the ability to handle both continuous and categorical data. Along with regression and classification, (Song and Ying 2015) has discussed the use of decision trees for:

1. Feature Selection: In most use cases a lot of features provided in the data space are not relevant. The methods used in the decision trees can be leveraged to select the most relevant features.
2. Identifying related importance between the features: Once the relevant features for a use case are identified. The relative importance of a feature can be decided by eliminating feature and checking the affect on model accuracy or purity of the splits. The more is the amount of loss in accuracy, the greater is the assessment of the importance of the feature.
3. Handling missing values: (Song and Ying 2015) has argues that eliminating the data points with missing values may not the most efficient way to handle the missing data and can cause bias to be

introduced in the analysis. Instead, decision trees can be used to handle the data by classifying the missing values to a separate category.

4. Data Manipulation: Some data mining applications deal with highly skewed continuous data or with categorical data with too many categories. Decision trees can help in reducing the number of categories and to divide the continuous feature values to ranges.

There are several algorithms that exist for implementing decision trees, including ID3 (Iterative Dichotomiser 3) (J. Ross Quinlan 1986) , C4.5 (J. Ross Quinlan 2014), CART (Classification and Regression Trees) (Breiman et al. 1984), CHAID (Chi-Squared Automatic Interaction Detection) (Kass 1980) and QUEST (Quick, Unbiased, Efficient, Statistical Tree) (Loh and Shih 1997) among others. Particulars of the various algorithms are discussed below:

```
## # A tibble: 5 x 5
##   Methods                              CART         C4.5      CHAID   QUEST
##   <chr>                                <chr>        <chr>     <chr>   <chr>
## 1 Feature Selection to create splits   Gini Index   Entropy/~ Chi-sq~ Cont. Data~
## 2 Classification/Regression            both         both      Classi~ Classifica~
## 3 Input feature type                   Cont./categ. Cont./ca~ Cont./~ Cont./cate~
## 4 number of splits at a decision node  2            multiple  multip~ 2
## 5 Pruning                              Pre-pruning  Pre-prun~ Pre-pr~ Post-pruni~
```

3 Methods

3.1 The basic principles

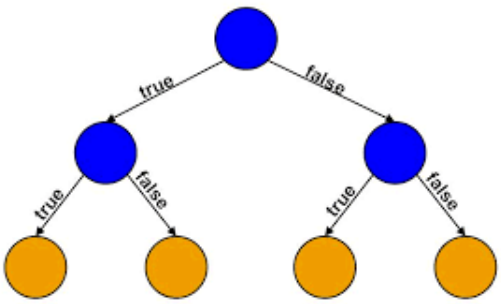


Figure 1: An example of decision tree.

A decision tree is majorly formed by nodes and branches. Nodes can be broadly categorized into two types. A decision node can either be a root node or an internal node in the decision tree. It presents a condition based on which the data space is split into two or more subsets. A leaf node is a node which has no sub-nodes. Hence, it is also called an end node. It represents the final prediction/classification made by the decision tree.

Branches of the tree indicate the possibilities of the splits from a decision tree. The decision tree is completely formed by the hierarchy of nodes and branches. The connection of branches between the root node and a leaf node forms a classification rule. This means, we start at the root node and move through the decision tree until a leaf node is encountered. At each decision node, depending on the splitting criteria, we choose the right branch leading us to the root of the sub tree. The process is repeated until we reach a leaf node where prediction of the class takes place. The structure of the decision tree is usually considered analogous to an 'if-then-else' conditional block of a programming language.

The objective of a decision tree classifier is to segregate the data space consisting of data points that belong to a mixture of classes into splits that contain data points belonging to a single class or a split where most of the data points belong to a single class, in case, a clean split cannot be made. If *train* denotes the training data entering into a node N A generic recursive algorithm to build a decision tree can be (Tan, Steinbach, and Kumar 2016):

1. If all the data points in *train* belong to a single class then N becomes a leaf node.
2. If *train* is an empty set, then N becomes a leaf node predicting a default class.
3. If *train* contains data points which belong to more than one class then a feature test is used to calculate the splitting criteria to split the data in the smaller subsets.

The steps are repeated until all records in a subset belong to or are predicted in a same class. However, care must be taken for a case where training data comprises of points where the feature values are all same but the associated classes are different. In such a case, splitting the data might not be possible and the node is declared as a leaf node and the predicted class is the most frequent class in the data set.

Thus, the splitting criteria becomes important for forming an efficient decision tree. The splitting criteria is calculated based on the values of various features passed as input to the decision tree.

3.2 Determining number of splits

There can be multiple ways to decide the number of subsets the data is split into by a split condition at a decision node. The type of the feature in consideration can be considered to decide on the number of splits formed:

1. Binary Attributes: The condition for binary attributes can be formed in such a way that the data is divided into two simple subsets, each pertaining to a category of the attribute. 2 illustrates an example of such a binary splitting.

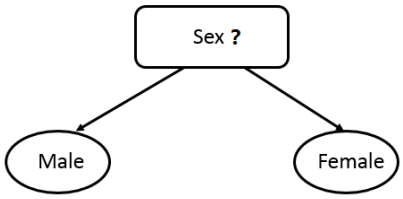


Figure 2: Splitting binary categorical data.

2. Categorical Data: For categorical features with more than two categories, one can decide on both binary splits where one subset consists of data points corresponding to just one category while the second subset contains data points corresponding to all other categories for the categorical feature, or multiple splits where each split corresponds contains data points corresponding to a single category corresponding to the categorical variable.

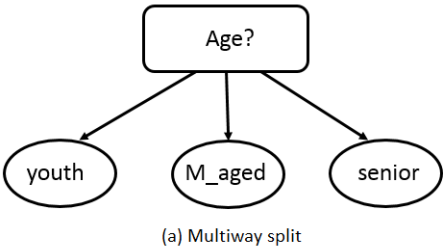


Figure 3: Multiple splitting for categorical data with more than 2 categories.

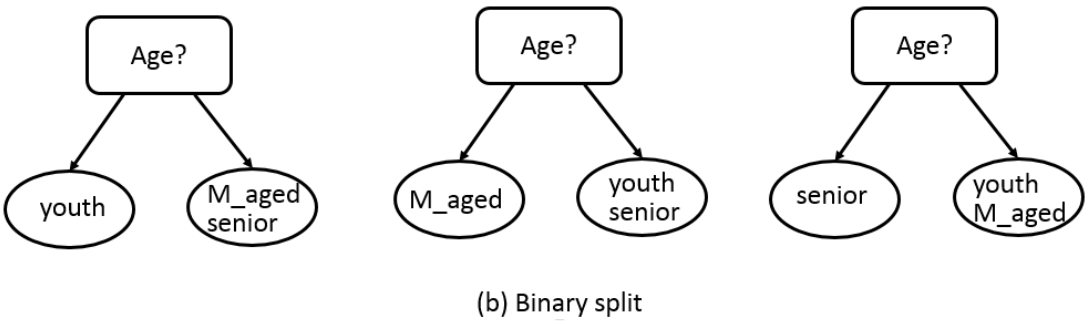


Figure 4: Binary splitting for categorical data with more than 2 categories.

3. Continuous Attributes. For continuous attributes, the test condition can be constructed as a comparison test $(Attribute <= V) or (Attribute > V)$ for getting binary splits, or a range query with outcomes of the form $(V_i <= Attribute < V_{i+1}, \forall i = 1, 2, 3, \ldots, m)$. The difference between these approaches is shown in 5 . For the binary case, the decision tree algorithm must consider all possible split positions V, and it selects the one that produces the best partition. Usually the split positions are taken to be the mid-points between all pairs of unique, subsequent values of a feature. For the multiple splits, the algorithm must consider all possible ranges of continuous values of the feature (Li 2021).

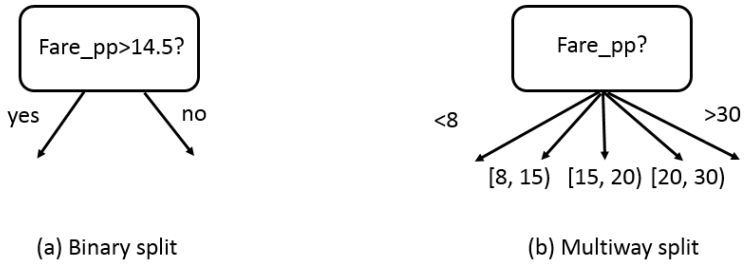


Figure 5: Splitting for continuous data.

3.3 Determining best split condition

Different decision tree algorithms utilize different methods to determine the best split criteria. When building the decision tree, the most important features are considered to split the data set at the root node and at further subsequent internal nodes into two or more splits. The methods that determine the splitting take into account the degree of 'purity' of the resultant child nodes. Some of these methods include: entropy (Russell and Norvig 2002), Gini index (Mohri, Rostamizadeh, and Talwalkar 2018), twoing criteria (Patel and Upadhyay 2012), information gain (Shalev-Shwartz and Ben-David 2014), classification error (Tan, Steinbach, and Kumar 2006) and gain ratio (J. Ross Quinlan 2014). When the most important features are identified corresponding to every unique feature value (for categorical data) or mid-point between two subsequent feature value we check for the purity of the split based on the method. And the best feature as well as the best split value is then used to create the split.

1. Gini Index: Gini index is also sometimes known as Gini impurity measures the probability of a datapoint being misclassified. The value of gini index varies between 0 and 1 where 0 indicates

that all the data points belong to the same class or only a single class exists while 1 indicates that the data points are randomly distributes in the data set being considered. A Gini index of 0.5 denotes equally distributed elements into some classes.

$$GINI(dataset) = 1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability of the object being classified to a particular class and n is the number of data points in the data set. While,

$$GINI(afterSplit) = \frac{|left|}{totalData} GINI(left) + \frac{|right|}{totalData} GINI(right)$$

2. Entropy: It is a measure of degree of impurity, uncertainty or randomness in a data set. It is usually clubbed with Information Gain ratio to decide on the best split. The formula for rntropy is given as:

$$E(x) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

3. Classification Error: It is a measure of the misclassified data points. The formula of the same is given by:

$$ClassificationError(x) = 1 - max(p_i)$$

One of the major disadvantages of the decision tree is is that for large data sets, the number of splits created by the decision tree increases and so does the depth of decision tree. Such complex architecture can cause the model to overfit as the model becomes too specific to the training data. Thus, upon seeing the new test data, the model might not generalize well hinders the model performance. Ideally, the decision tree grows until the data points in each leaf node belong to a single class only. In the worst case the leaf node might have only a single data point which means the model might not be able to reliable make predictions. In order to prevent this behaviour, certain stopping criteria need to be applied which prevent the tree from growing overly complex. Implementing these criteria is known as pruning the decision tree. Pruning can be applied while building the tree, which is termed as pre-pruning, or can be applied after the tree is already built.
Some of the pre-pruning techniques include:

1. Regulating the max depth of the decision tree.
2. Controlling the minumum number of datapoints in a leaf node.
3. Determining the number of datapoints in the decision node to decide on further splitting.

(Berry and Linoff 2000) recommend the number of data points in the leaf node to be restricted to 0.25% to 1% of the total data points in the data set.
Post pruning allows the decision tree to grow and then prunes is to optimal size by replacing decision nodes that do not provide any additional information with leaf nodes (Friedman et al. 2001).

The leaf nodes formed after pruning the tree may not be completely pure and may contain data points associated with multiple classes. The predicted class is thus, the most frequent class present in the leaf node.

3.4 Functional specification of the implemented decision tree

A decision tree is implemented from scratch as a part of this report. The implemented decision uses a class implementation to store the state of the decision tree. The major functionalities of the implementation are:

1. **build_tree(dataset, header=None, split_type='gini,' min_samples_split=2, max_depth=None, counter=0,featureType=None):**
   This is the major function that builds the decision tree using recursion. The **dataset** passed is a list of lists with labels appended as a last attribute of the data point.
   The **header** is a list of names of the features which is passed separately. If not passed, then instead of names feature indices are used in the algorithm.
   split_type determines the splitting criteria. The implementation supports gini impurity and entropy type. The current implementation does not support infirmation gain.
   min_samples_split specifies the number of data points that must be present in the dataset to continue further splitting at a decision node.
   max_depth helps in restricting the size of the decision tree to a certain depth. **counter** helps in ensuring the maximum depth restriction.
   featureType is an optional list that can be passed along with the data set to specify the type of the features in the data set. This helps in controlling the split criteria for continous and categorical features. If not passed the the method calls **get_feature_type** to specify the type of the features based on certain defaults. The feature types are stored globally for further use. Thus, along with continous data, categorical data is also handled in this implementation. For every decision node, the class implementation helps in storing the split value, the feature on which the split value was obtained and the operation (corresponding to the type of the feature) is stored to form the split condition. This helps in making predictions after the tree is formed/trained. For every leaf node, a prediction is stored in the state of the node. The function returns the root of the decision tree formed.

2. **find_best_split(dataset, split_type):**
   Depending on the **split_type**, this function helps in determining the best feature and corresponding best split-point to split the **dataset**. Only two splits are created at every decision node. Thus, this function traverses through every possible split of every feature tries to find The split point corresponding to which the minmum entropy or gini index is observed is used to split the data. Base cases are provided for any possible errors which are handled in the build_tree function. It returns the minimum gini index/entropy for the given data set, the split value at which the minimum gini/entropy was achieved, the feature index for which the split value was achieved along with the splitted data.

3. **find_centres(dataset):**
   This function is used to find all the potential split points against all the features of the **dataset** passed. For continous variables the split-points are the mid-points between every pair of unique subsequent feature values, while for the categorical features, the splitpoints are categories. 5 illustrates the splitting utilized for categorical features.It returns a dictionary where each key corresponds to a feature in the data set while the value against it is the list of all potential split points.

4. **split_data(dataset, feature_index, centre):**
   This function is used to split the **dataset** in two sub sets based on the **centre** and the **feature_index**. For continous feature, the left split will contain all the data points where the corresponding feature value is <= the centre and the rest are stored in the right split. For categorical data, the left split contains the data points where the corresponding feature value is equal to the centre while the rest are stored in the right split. The empty left or right splits are handled in the **build_tree** function. It returns the splitted dataset.

5. **entropy(dataset), gini(dataset):** These functions are used to create the entropy and the gini index for the **dataset** passed. While building the decision tree these functions are utilized to calculate the entropy and gini values for each split separately.

6. **entropy_split(left, right), gini_split(left, right):** These functions are utilized to find the overall entropy and the gini index (respectively) for the data set after splitting. In order to do so, they internally call **entropy and gini** functions.

7. **get_feature_type(dataset, threshold=10):**
   This function helps the decision tree to handle categorical data. It helps in establishing the type of various features passed in the **dataset** as continuous or categorical. If a feature contains string

values then it is regarded as a categorical feature. For features with integer or float values, if number of unique values in the feature are less than the **threshold** then the feature is regarded as a categorical feature else, the feature is assumed to be continuous. Depending on the use case, it is advised to manually call this function to determine the feature type and then pass the list returned in the **build_tree** function. The final output of this function is a list holding type of the feature. The index of this list corresponds to the feature index.

8. **predict_(root, test):**
   This function is single used to get the prediction against the **test**, which is a single datapoint. It recursively checks the value stored against each node of the tree. For the leaf node, the prediction is stored as a string, while for a decision node the value contains a list of operator and the split value. Based on the operator and the split value the algorithm tries to reach a leaf node and returns the predicted label.
   root is the root of the decision tree returned by **build_tree** function.

9. **predict_single(root, sample):**
   This function is used to make predictions for a **sample** of data containing one or more than one data points. It internally calls **predict** function to return predictions against each data point in the sample. It returns a list of predicted labels against each data point.

10. **print_tree(root, spacing=" "):**
    This recursive function uses in order traversal of the decision tree to print the tree structure. The **spacing** helps in appending horizontal whitespace which is multiplied as the depth of the tree increases. Thus, nodes at the same depth have same horizontal whitespace before them.

Other helperfunctions are also implemented in the source code shared along with this report.

3.5 Libraries and their specifications

1. **Scikit-learn** *version- 0.24.2:* It is a popular machine learning library for python programming language, which consists of various clustering, classification and regression algorithms along with implementation of serveral performance metrics to compare the results. The library also contains modules for data preprocessing "1. Supervised Learning" (2021).
   The performance of the decision tree classifier implemented is compared against the scikit-learn's library implementation of decision tree. The performance of the models are the compared against performance metrics like accuracy, precision, recall and F1 score which are implemented in the scikit-learn library.

2. **Pandas** *version- 1.3.4:* It is an open library which provides poweful and flexible constructs for source data analysis and manipulation. It used to transfer the results to a csv file.

3. **memory_profiler** *version- 0.58.0* It used for monitoring the memory consumption of a function. The function memory_usage of the library is used to compare the memory consumption of the two implementations of the decision tree. It returns a list of scalar values which give an estimate of the memory consumption. The maximum value in the list is used for comparison

4. **time:** It is used to compute the time performance of the two implementations of the decision tree classifiers. The perf_counter function of the library is used to measure the time taken by the tree to train.

5. **random:** It is used to split the data into training and testing data.

6. **math:** it is used to calculate the log of the probability of a data point belonging to a particular class while calculation of entropy.

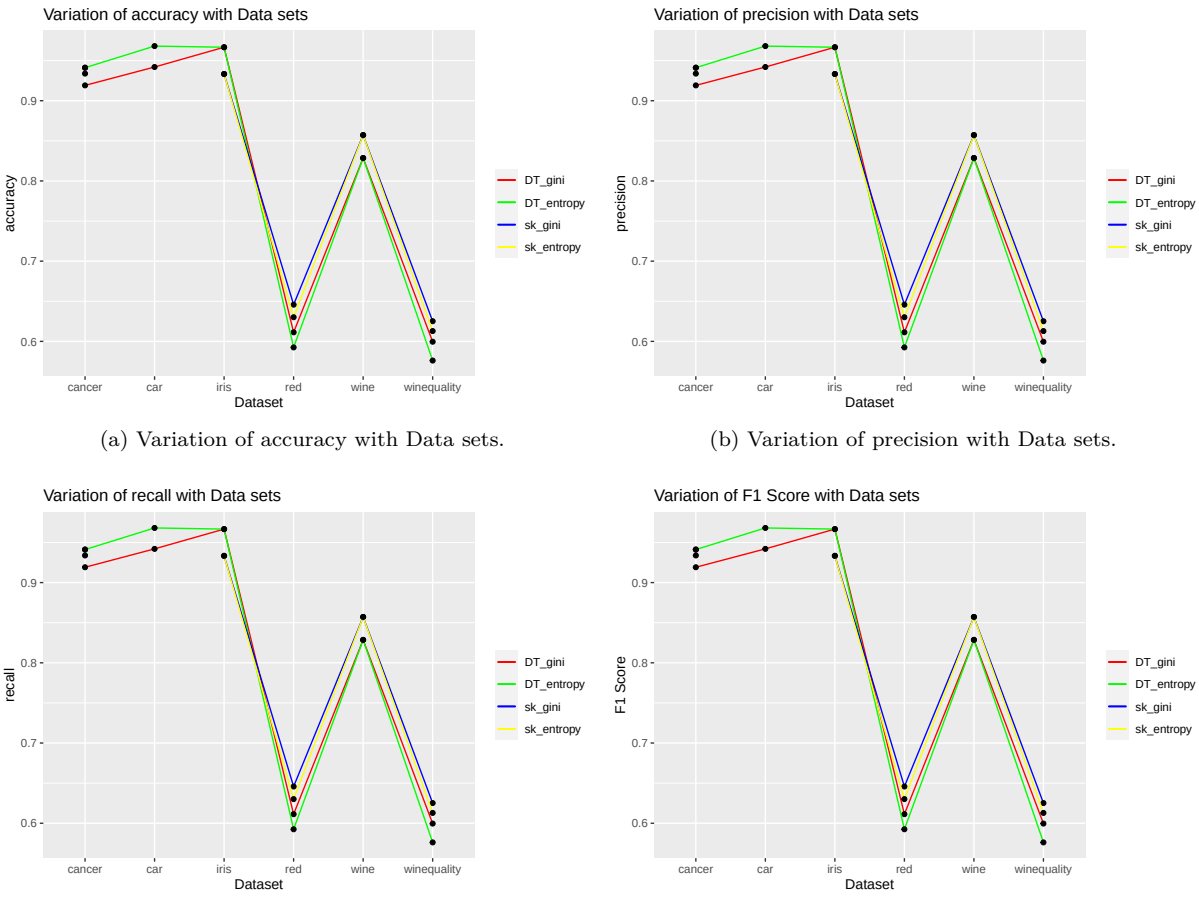
4 Results

The performance of both the implementation of the decision trees on various data sets, considering both gini idex and entropy as splitting criteria, is measured using the metrics - time, memory, accuracy, precision, recall and F1 score. The results obtained are discussed below.

4.1 Handling categorical data

It is observed that scikit-learn implementation of the decision tree cannot handle categorical data directly. If the feature values are strings, the decision tree throws the error directly. Converting the categories into integers using label encoding enables the program to handle the data, but the features are treated as continuous features only. Thus, the splits generated are not semantically correct. Whereas, the implementation of the decision tree discussed in this report can handle categorical data with both string or integer values.

4.2 Performance metric values for various data sets



(a) Variation of accuracy with Data sets.

(b) Variation of precision with Data sets.

(c) Variation of recall with Data sets.

(d) Variation of F1 score with Data sets.

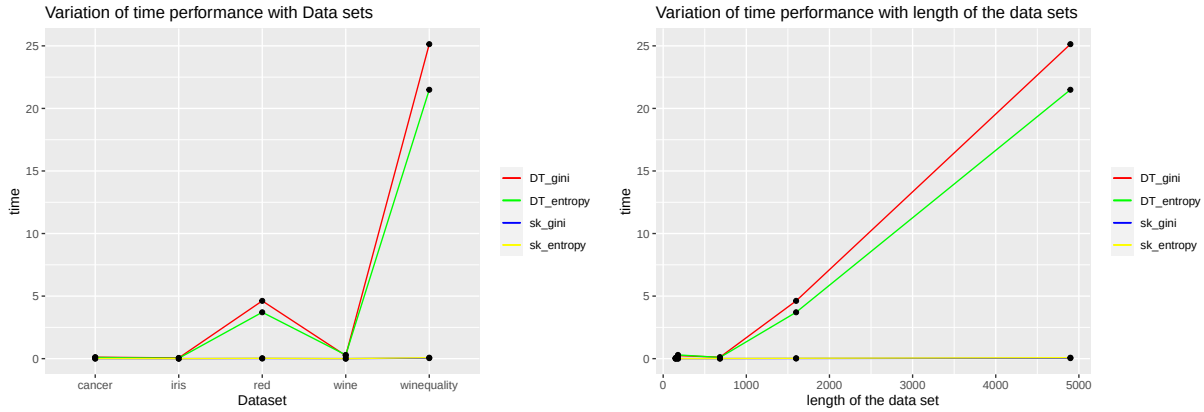
Figure 6: Variation of accuracy with Data sets.

The performance in terms of accuracy is found to be comparable for both the implementations across both gini and entropy splitting criteria, which is illustrated in 6-(a). Similar results are observed for precision, recall and F1 score which is illustrated in 6

In time performance, the scikit-learn implementation is found to be much better than the proposed decision tree (7). This could be because the constructs of the scikit-learn are implemented using cython wrappers which them faster than the core python programming language. Linear regression models were fitted to extrapolate the time performance of the decision tree implementations on very large data sets.

4.2.1 linear regression for proposed implementation of decision tree with gini index.

```
##
## Call:
```



(a) Variation of time performance with Data sets. (b) Variation of time performance with length of the data sets

Figure 7: Time performance of the decision tree implementations

```
## lm(formula = my_time ~ length, data = cont_data)
##
## Residuals:
##       1       2       3       4       5
##  1.3262  0.7527 -1.9425  1.3595 -1.4958
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.0775077  1.0787198  -1.926  0.14977
## length       0.0054024  0.0004636  11.653  0.00136 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.843 on 3 degrees of freedom
## Multiple R-squared:  0.9784, Adjusted R-squared:  0.9712
## F-statistic: 135.8 on 1 and 3 DF,  p-value: 0.001358
```

4.2.2 linear regression for proposed implementation of decision tree with entropy.

```
##
## Call:
## lm(formula = my_time_ent ~ length, data = cont_data)
##
## Residuals:
##       1       2       3       4       5
##  1.1395  0.7033 -1.8681  1.2787 -1.2534
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.8002815  0.9848458  -1.828  0.16500
## length       0.0046120  0.0004233  10.896  0.00165 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.682 on 3 degrees of freedom
## Multiple R-squared:  0.9754, Adjusted R-squared:  0.9671
## F-statistic: 118.7 on 1 and 3 DF,  p-value: 0.001655
```

4.2.3 linear regression for the library implementation of decision tree with gini index.

```
##
## Call:
## lm(formula = sk_time_ent ~ length, data = cont_data)
##
## Residuals:
##          1          2          3          4          5
## -0.0012961 -0.0008603  0.0030522 -0.0002764 -0.0006194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.878e-03  1.180e-03   1.591     0.21
## length      1.576e-05  5.074e-07  31.054 7.34e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002016 on 3 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9959
## F-statistic: 964.4 on 1 and 3 DF,  p-value: 7.336e-05
```

4.2.4 linear regression for the library implementation of decision tree with entropy.

```
##
## Call:
## lm(formula = sk_time_ent ~ length, data = cont_data)
##
## Residuals:
##          1          2          3          4          5
## -0.0012961 -0.0008603  0.0030522 -0.0002764 -0.0006194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.878e-03  1.180e-03   1.591     0.21
## length      1.576e-05  5.074e-07  31.054 7.34e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002016 on 3 degrees of freedom
## Multiple R-squared:  0.9969, Adjusted R-squared:  0.9959
## F-statistic: 964.4 on 1 and 3 DF,  p-value: 7.336e-05
```

p-value against length < 0.05 for all the models. Therefore, the variable is significant. The result of the regression is shown in 8. It is confirmed by the regression analysis that the library implementation of the decision tree is significantly better. For the proposed model, it is observed that when using entropy, the time required to train the model is a little less than when gini index is being used.

The performance of both the implementations of decision tree are found to be comparable in terms of memory performance, with proposed implementation consuming slightly lesser memory for larger data sets than scikit-learn's decision tree (9).

4.3 Formal comparision of the models

In order to formally compare the the two implementations of the decision trees, Wilcoxon signed rank test (Wilcoxon 1992) is utilized as it does not require any assumption of Normal distribution of the variables being compared. Moreover, the values of performance metrics for both the models were taken on the same data sets. Thus, the conditions to perform the test are satisfied. As values for accuracy,

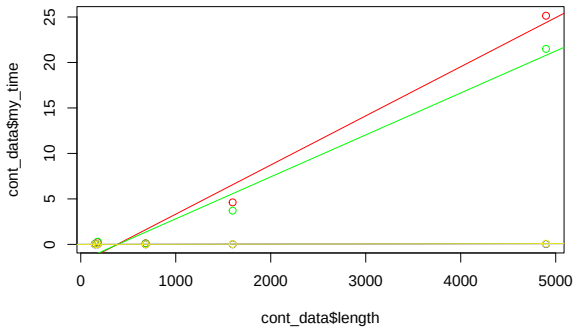
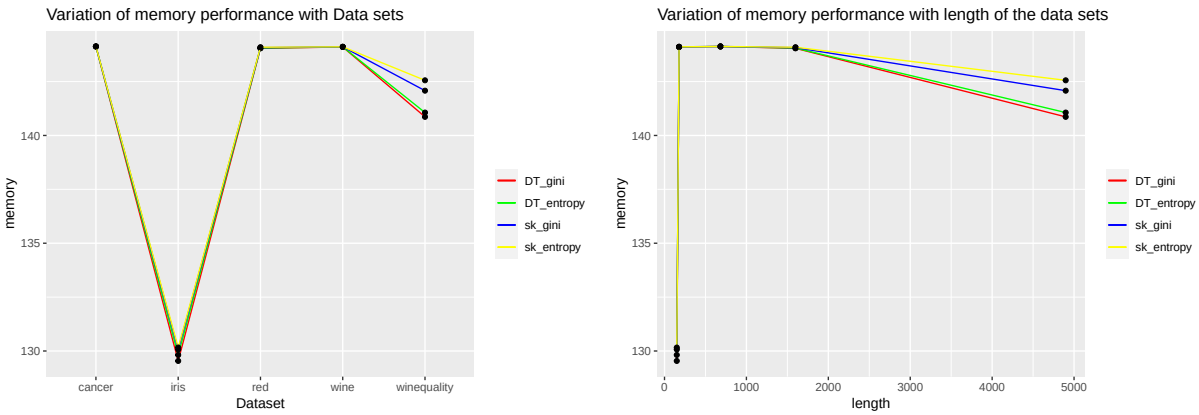


Figure 8: Analysis of linear regression on the time perofrmance of the decision trees.



(a) Variation of memory performance with Data sets.

(b) Variation of memory performance with length of the data sets

Figure 9: Memory performance of the decision tree implementations

precision, recall and F1 score have similar distributions in the observed sample, the non-parametric test is applied on the observed accuracy values. The test assumes the null hypothesis that the distribution of accuracy for both the models is similar which makes the alternate hypothesis that the distribution of accuracy for the models is not similar. Similar dsitribution of accuracy would mean that the performance of the models is comparable.

4.3.1 Comparison of proposed implementation against library implementation with gini

```
##
##  Wilcoxon signed rank exact test
##
## data:  cont_data$my_accuracy and cont_data$sk_accuracy
## V = 4, p-value = 0.4375
## alternative hypothesis: true location shift is not equal to 0
```

4.3.2 Comparison of proposed implementation against library implementation with entropy

```
## Warning in wilcox.test.default(cont_data$my_accuracy_ent,
## cont_data$sk_accuracy_ent, : cannot compute exact p-value with zeroes

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  cont_data$my_accuracy_ent and cont_data$sk_accuracy_ent
## V = 2, p-value = 0.3613
## alternative hypothesis: true location shift is not equal to 0
```

4.3.3 Comparison of proposed decsion tree with gini against proposed decsion tree with entropy

```
## Warning in wilcox.test.default(cont_data$my_accuracy,
## cont_data$my_accuracy_ent, : cannot compute exact p-value with zeroes

##
##  Wilcoxon signed rank test with continuity correction
##
## data:  cont_data$my_accuracy and cont_data$my_accuracy_ent
## V = 4, p-value = 0.7893
## alternative hypothesis: true location shift is not equal to 0
```

It is observed that distribution of accuracy for the models compared are not significantly different from each other as the p-values came out to be much larger than the 0.05 significance level. Thus, we cannot reject the null hypothesis that there is a significant difference in performance of the models. The only difference is observed to be in the time performance of the models.

4.4 Affect of pruning the decision tree

The performance of the proposed decision tree was observed on changing the maximum depth of the decision tree and the minimum number of samples required to split a decision node for a sample data set. It is inferred that decision trees can easily overfit and the performance of the model can be improved my controlling the mentioned stopping criteria. Upon reducing the depth of the decision tree the model generalizes better and an increase in accuracy and other metrics is observed.

Upon increasing the minimum number of samples to be present in the data set to split further, the model begins to generalize better resulting in an increase performance metric values.

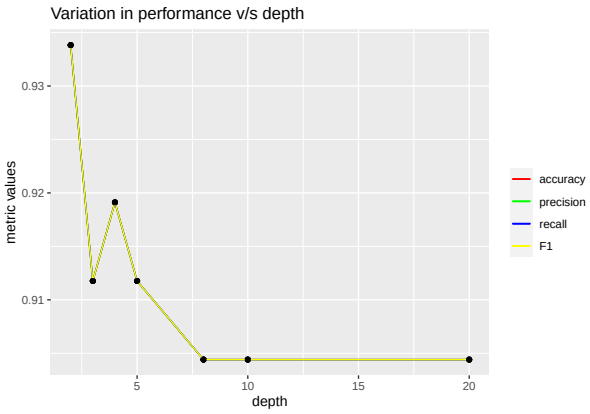


Figure 10: pruning maximum depth.

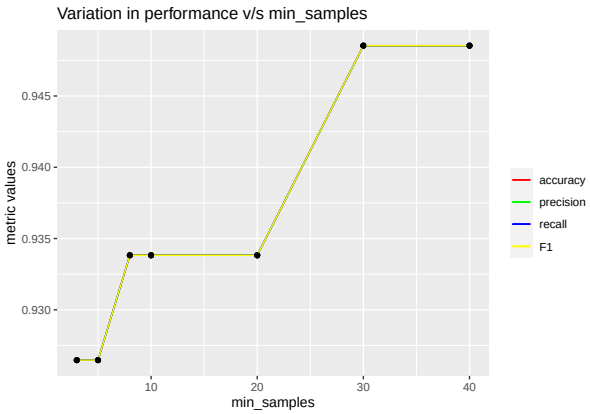


Figure 11: pruning minimum samples to split a node.

5 Conclusion

The underlying concept of the decision tree is discussed in detail and an implementation of the decision tree based on the same is proposed. The proposed decision tree was compared to the scikit-learn's library implementation of the decision tree classifier based on performance metrics like accuracy, precision, recall, F1, time performance and memory consumption and the observed results were discussed as well. The affect of pruning the decision tree were also analyzed. Performance comparison based on gini index and entropy was discussed

Further work can be done in reducing the time complexity of the proposed model. Information gain as a splitting criteria can also be incorporated in the proposed model. Implementation of post-pruning techniques and other stopping criteria can also be evaluated.

References

"1. Supervised Learning." 2021. *Scikit*. https://scikit-learn.org/stable/supervised_learning.html.

Berry, Michael A, and Gordon S Linoff. 2000. "Mastering Data Mining: The Art and Science of Customer Relationship Management." *Industrial Management & Data Systems*.

Breiman, Leo, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and Regression Trees*. CRC press.

Friedman, Jerome, Trevor Hastie, Robert Tibshirani, et al. 2001. *The Elements of Statistical Learning*. Vol. 1. 10. Springer series in statistics New York.

Jain, Anil K, M Narasimha Murty, and Patrick J Flynn. 1999. "Data Clustering: A Review." *ACM Computing Surveys (CSUR)* 31 (3): 264–323.

Kass, Gordon V. 1980. "An Exploratory Technique for Investigating Large Quantities of Categorical Data." *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 29 (2): 119–27.

Li, Gangmin. 2021. "Do a Data Science Project in 10 Days." *What Is This Book?* https://bookdown.org/gmli64/do_a_data_science_project_in_10_days/what-is-this-book.html.

Loh, Wei-Yin, and Yu-Shan Shih. 1997. "Split Selection Methods for Classification Trees." *Statistica Sinica*, 815–40.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. *Foundations of Machine Learning*. MIT press.

Murty, M Narasimha, and V Susheela Devi. 2015. *Introduction to Pattern Recognition and Machine Learning*. Vol. 5. World Scientific.

Ng, Andrew Y, and Michael I Jordan. 2002. "On Discriminative Vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes." In *Advances in Neural Information Processing Systems*, 841–48.

Patel, Nikita, and Saurabh Upadhyay. 2012. "Study of Various Decision Tree Pruning Methods with Their Empirical Comparison in WEKA." *International Journal of Computer Applications* 60 (12).

Quinlan, J Ross. 2014. *C4. 5: Programs for Machine Learning*. Elsevier.

Quinlan, J. Ross. 1986. "Induction of Decision Trees." *Machine Learning* 1 (1): 81–106.

Russell, Stuart, and Peter Norvig. 2002. "Artificial Intelligence: A Modern Approach."

Shalev-Shwartz, Shai, and Shai Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press.

Song, Yan-Yan, and LU Ying. 2015. "Decision Tree Methods: Applications for Classification and Prediction." *Shanghai Archives of Psychiatry* 27 (2): 130.

Tan, Pang-Ning, Michael Steinbach, and Vipin Kumar. 2006. "Data Mining Introduction." Bei Jing: The people post; Telecommunications Press.

———. 2016. *Introduction to Data Mining*. Pearson Education India.

Wilcoxon, Frank. 1992. "Individual Comparisons by Ranking Methods." In *Breakthroughs in Statistics*, 196–202. Springer.