Development and Analysis of Decision Tree Classifier

Kyriakos Chiotis (35278597)

Abstract—**This document analyses the implementation and evaluation of the decision tree classifier. The classifier is developed with the Python programming language and its evaluation is achieved by comparing it with the DecisionTreeClassifier class of scikit-learn library[1]. The evaluation is applied on two different data-sets of UCI Machine Learning Repository(Iris[2] and Mushroom[3] data-sets) in order to handle and test both numerical and categorical features. After repeatedly collecting various metrics and changing the max depth, the implemented and scikit-learn algorithms are analysed and compared using R. The final conclusion is that both algorithms operate similarly in case of numerical data. However, in case of categorical data there is different approach while decision tree implemented in scikit-learn uses only numerical features and these features are interpreted always as continuous numeric variables. Therefore, all categorical features are converted to dummy variables in order to compare the two algorithmic approaches. The objectives and tasks of this report are based on the final coursework of module SCC461: Programming for Data Scientists of Lancaster University.**

I. Introduction

Decision tree algorithm is one of the methods of predictive modeling used in statistics, data mining and machine learning. As the name goes, it uses a tree-like model of decisions to move through branches from observations on an item to conclusions on the target value of the item. The target variable may consist of a discrete set of values or continuous numerical values covering classification and regression purposes. The observations answer binary questions in order to split the data and reduce the predictive uncertainty(or impurity) of the target value until no more reduction can be achieved. In data mining terminology, the first question and the final observations represent the root and leaf nodes, respectively.

Decision trees are simple to understand and interpret providing visualization of the learning process. Building a decision tree is an NP-complete problem which means that splits are locally optimal fluctuating the depth between O(logN) and O(N), where N is the number of the samples. The time complexity is O(Nkd), where k is the number of the features and d is the depth of the tree. Consequently, the best and worst case of training a decision tree is accomplished in O(NlogN) and O(N^2), respectively.

The main and most challenging objective of a decision tree classifier is to split the data by measuring the impurity of the data. There are a bunch of ways to measure impurity such as Gini Index, Entropy or Chi square. Therefore, the implemented training algorithm is a recursive algorithm who stops when the maximum depth is reached or no more impurity decrease can be achieved. This recursive algorithm is called CART, which means Classification And Regression Trees[4]. This document is focusing on classification problem with Gini Index as impurity measure. Finally the research strategy is formulated as follows:

1) Recap of decision tree algorithm
2) Selection and pre-processing of data-set
3) Development of decision tree algorithm
4) Analysis and comparison with original algorithm

II. Methods

A. Recap of decision tree algorithm

First step in the methodology is the recap of decision tree algorithm. Decision tree is a tree based model which is very easy to implement and interpret it's output. In this algorithm the data is split into homogeneous sets based on splitting criteria with most popular the Gini Index. The creation of sub-nodes increases the homogeneity of resultant sub-nodes by decreasing the impurity of the node with respect to the target variable. The impurity of a node or the Gini can be calculated as follows:

$$G = 1 - \sum_{k=1}^{n} p_k^2 \tag{1}$$

where n is the number of training samples and p_k is the fraction of samples belonging to class k. The best splitting point is found by minimizing the weighted average Gini for the children nodes which is calculated as:

$$TotalGini = \frac{i}{m} G^{left} + \frac{m-i}{m} G^{right} \tag{2}$$

where m the elements on the parent node, i the elements on the left node and $m - i$ the elements on the right node. To conclude, the development of decision tree algorithm should be based on the following steps:

1) Iterate through the unique feature values and keep track the feature index and feature value with the lowest impurity decrease.
2) Split the data set and repeat step 1 for the children nodes.
3) Break when impurity decrease equals 0.

B. Selection and pre-processing of data-set

The selection of the data-sets is done in order to test and handle numerical and categorical features and binary and multi-class classification. The numerical features and multi-class classification are tested on Iris data set, whilst the categorical features and binary classification are tested on Mushroom data set.

The Iris data set is comprised of 4 numerical features and 3 classes of 50 instances each, where each class refers to a type of iris plant. The data are balanced and contain no missing values. Therefore, the pre-processing phase is dealing only with the separation of features and class labels. On the other hand, mushroom data are balanced but contain 2,480 missing values in the form of '?' at stalk-root feature. Therefore, the missing values are imputed with the most common value. Finally, feature selection based on feature importance of Random Forest reduces the dimensionality to 3 dimensions. This reduction makes the visualization and interpretation of decision tree simpler but the process of feature importance is not analysed in this paper.

C. Development of decision tree algorithm

The development of decision tree is implemented in Python language and uses object-oriented programming approach. Consequently, the first step is to set the classes which are "Node" class and "MyDecisionTree" class.

Class "Node" represents each node on the tree and contains all the relevant information. This information contain the data, a feature index, a feature value, the impurity of the node, the data for the left node, the data for the right node, a threshold and two pointers for the children nodes. Also, there is a method called "info_gain" which measures how much information is gained by splitting the data on this node. Specifically, this method returns a value calculated as:

$$info\_gain = Impurity - TotalGini \tag{3}$$

where $TotalGini$ is declared on equation (2). As it was said, decision tree algorithm seeks for the lowest impurity decrease. According to equations (2) and (3), the implemented algorithm is seeking for the highest information gain. The threshold parameter represents a minimum impurity decrease which means that if info_gain is less than the threshold, then info_gain equals zero.

Class "MyDecisionTree" contains two parameters which operate as thresholds. The first one is a maximum depth threshold and the second one is the aforementioned minimum impurity decrease threshold. In addition, class methods create the tree by fitting the training data set on the algorithm and also predict the class labels of the testing data according to the created tree. Therefore, the usage of the implemented tree could be described as:

1) Create a decision tree based on a training set.
    a) Iterate through the unique feature values and keep track the feature index and feature value with the highest information gain.
    b) Split the data set and repeat step 1 for the children nodes.
    c) Escape the iteration when information gain equals 0 or maximum depth is reached.
2) Predict the class label.
    a) Follow for each record the nodes with matched branches.
    b) If the following node is a leaf, predict the most common value.
3) Visualize and interpret the decision tree.

Last but not least, all the methods and classes are checked with unit test cases. Unit test cases are applied on a simple synthetic data set which covers most of the aspects of the classification method.

D. Analysis and comparison with original algorithm

The analysis and evaluation of the implemented algorithm is achieved by comparing its metrics with the metrics of scikit-learn decision tree algorithm. These metrics regard accuracy, precision, recall and computational time of fitting the training data. Also, the metrics are collected repeatedly using various depths and different training sets with 10-fold cross validation. Finally, tree visualization is used to compare the two trees.

After collecting the metrics, they are exported in R studio for further analysis. The data are analysed exploratory using graphical visualization of ggplot2 library. Therefore, t-tests are applied in order to confirm if both algorithms predict similarly the class labels of testing data. It is essential to stress out that the sets of observations are not independent as they concern a specific part of the data set and need to be handled as paired. Finally, Mushroom data set is used to compare the time complexity of the two algorithms. This is achieved by collecting the computational time of the fitting repeatedly on different sub data sets of Mushroom.

III. Results

A. Iris data set

The results on the Iris data set are compared initially by visualising the tree. It is observed that for certain training data Sklearn algorithm may create different trees.

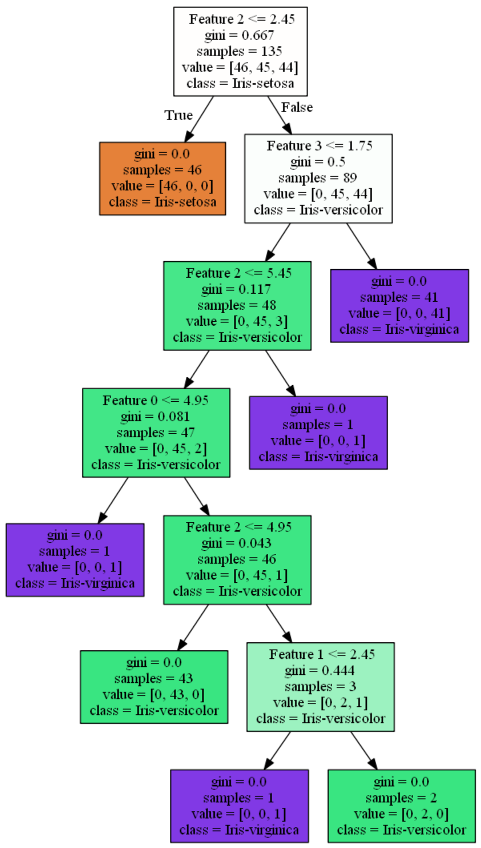

Fig. 1. Sklearn decision tree with root node on feature 2

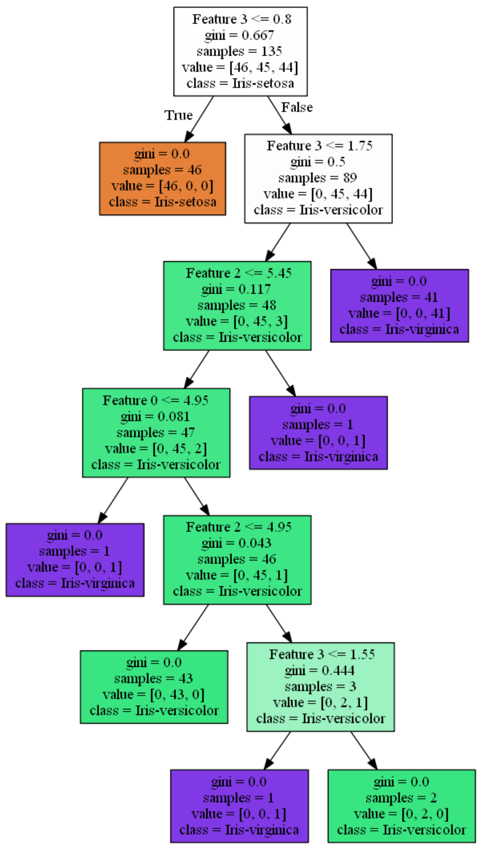

Fig. 2. Sklearn decision tree with root node on feature 3

This variety in Figure 1 and Figure 2 is based on the fact that if the data are split on feature 2 or on feature 3, provide the same impurity reduction. This is explained also by the class prediction for depth 1. In order to handle this phenomenon and have balanced metrics for comparison, the implemented algorithm may split on the first or the last best feature. Consequently, the average metrics rounded on two decimals are shown in Figure 3 below.

Depth	Algorithm	Accuracy	Precision	Recall	Time
1	Implemented	0.58	0.45	0.66	0.02
2	Implemented	0.94	0.94	0.94	0.02
3	Implemented	0.94	0.94	0.94	0.02
4	Implemented	0.94	0.94	0.94	0.02
5	Implemented	0.94	0.95	0.94	0.02
6	Implemented	0.95	0.95	0.95	0.03
1	Sklearn	0.58	0.46	0.66	0.00
2	Sklearn	0.94	0.94	0.94	0.00
3	Sklearn	0.94	0.94	0.95	0.00
4	Sklearn	0.94	0.95	0.95	0.00
5	Sklearn	0.95	0.95	0.95	0.00
6	Sklearn	0.95	0.95	0.95	0.00

Fig. 3. Table with the average metrics by depth and algorithm

Apparently, the averages of the two algorithms for each depth are very close. Therefore, boxplots of all metrics may be more useful to check any differentiation.

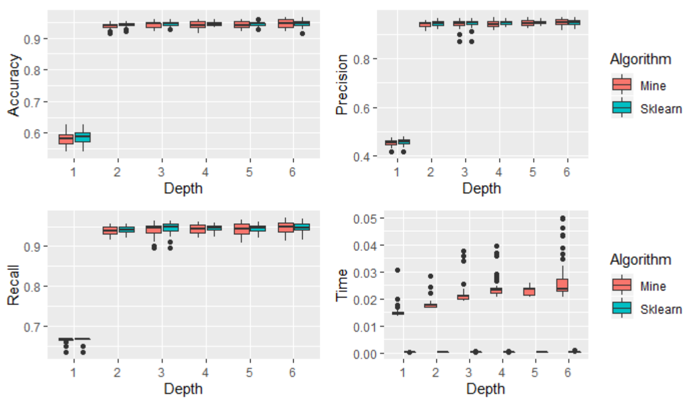

Fig. 4. Sklearn decision tree with root node on feature 3

In Figure 4 is obvious that both algorithms are making predictions with accuracy over 90% after depth one. The variance on implemented algorithm is slightly higher and the computational time is almost 60 times greater than Sklearn's algorithm. Finally, t-tests are applied in order to verify with 95% confidence that the predictive power of the two algorithms is the same at maximum depth(=6).

Hypothesis Studied	P-Value
H_0: Sklearn accuracy is equal to implemented accuracy H_1: Sklearn accuracy is different to implemented accuracy	1
H_0: Sklearn precision is equal to implemented precision H_1: Sklearn precision is different to implemented precision	0.8245
H_0: Sklearn precision is equal to implemented recall H_1: Sklearn precision is different to implemented recall	0.8653

Fig. 5. Table of t-tests

In Figure 5 is obvious that p-values are way greater than 0.05 which means that there is no evidence to reject the null hypothesis and we conclude that both algorithms have the same predictive power.

B. Mushroom data set

In order to compare the two algorithms on the Mushroom data set, the features have to be encoded with one hot encoding. This method converts the categorical features to dummy variables and they are handled as numerical. The metrics of both algorithms are summarised as:

	Sklearn	Implemented
Accuracy	0.9689	0.9689
Precision	0.9308	0.9308
Recall	0.9746	0.9746
Time	0.0011	0.0674

Fig. 6. Table of metrics

The table in Figure 6 shows that both algorithms handle similarly the binary classification task. However, the computational time of the implemented algorithm is much higher. The tree that is constructed is visualized as:

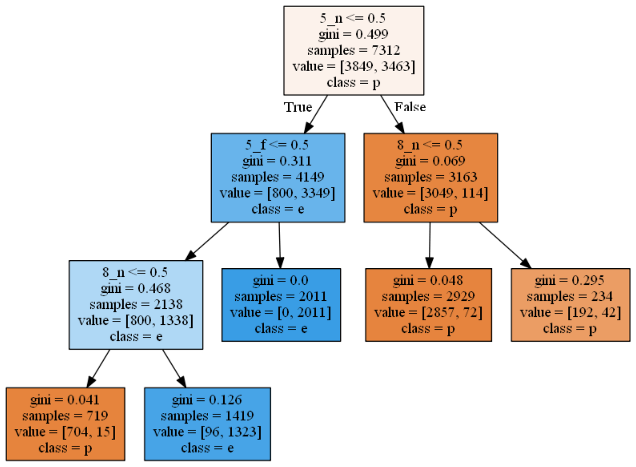

Fig. 7. Mushrooms decision tree

Classification predictive power is fine in both algorithms. However, there is quite big difference in fitting time. This difference does not mean something for the complexity of the implemented algorithm. A way to calculate the complexity is to collect time metrics with different sizes of data set. Therefore, linear regression and box-cox transformation are applied in order to check the time complexity. This analysis is demonstrated below:

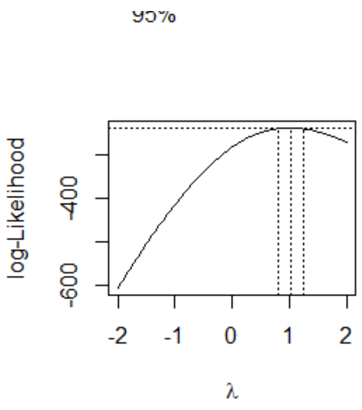

Fig. 8. Boxcox plot for Sklearn algorithm

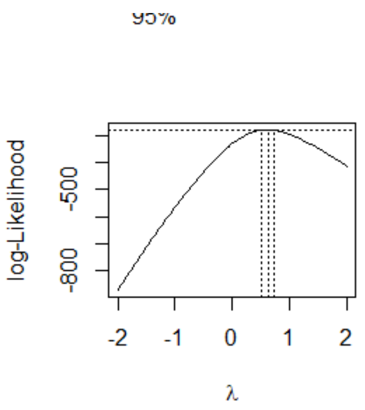

Fig. 9. Box-cox plot for Implemented algorithm

Figure 8 shows that no transformation is required because lambda is close to 1. Therefore, it could be said that the time complexity of Sklearn is O(n). On the other hand, in Figure 9, it is obvious that box-cox method is recommending a transformation with lambda close to 0.6. Consequently, the complexity of the implemented algorithm is a little less than O(n^2).

IV. Discussion

In general, the analysis and testing on the two data sets provide evidence that the implemented decision tree classifier is behaving in similar way with scikit-learn library. Through the research the original algorithm is operating with a nature of randomness. The random states regard different tree constructions with respect to same splitting criteria(i.e. minimum impurity decrease). The repetitiveness of the algorithm could demonstrate feature importance. Therefore, an upcoming update of the implemented algorithm could be in best split function. Specifically, it could store all the possible best splits and randomly choose one. This utility would reduce the bias of the results.

Last but not least, the implemented algorithm has worse time complexity than the original. However, R analysis indicated that it is between the worst and best case time complexity. Nevertheless, it could be verified that the implemented algorithm could be improved.

References

[1] scikit-learn DecisionTreeClassifier
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[2] Iris Data Set
http://archive.ics.uci.edu/ml/datasets/Iris
[3] Mushroom Data Set
http://archive.ics.uci.edu/ml/datasets/Mushroom
[4] Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole.