

FRANKFURT UNIVERSIY OF APPLIED SCIENCES, MARCH 2019, AI/ML SOFTWARE ENGINEERING SUPERVISED BY DAMIR DOBRIC

# Supervised learning with Decision Trees

Sai Sudha Murali

*Abstract*

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. DTs are considered to be one of the most popular approaches for representing classifiers.

The goal is additionally to do that in C# netcore for learning api.

[Too](file:///\\\\\\\Too) short

Keywords — Decision tree, Information Gain, Gini Index

1. **INTRODUCTION**

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal or decision nodes). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attribute’s value.

Each leaf is assigned to one class representing the most appropriate target value. Alternatively, the leaf may hold a probability vector indicating the probability of the target attribute having a certain value. Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path. Usually the tree complexity is measured by one of the following metrics: the total number of nodes, total number of leaves, tree depth and number of attributes used. Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part and taking the leaf’s class prediction as the class value.

1. **ALGORITHMIC FRAMEWORK FOR DECISION TREES**

Decision tree inducers are algorithms that automatically construct a decision tree from a given dataset. Typically the goal is to find the optimal decision tree by minimizing the generalization error. However, other target functions can be also defined, for instance, minimizing the number of nodes or minimizing the average depth. Induction of an optimal decision tree from a given data is considered to be a hard task. It has been shown that finding a minimal decision tree consistent with the training set is NP–hard (Hancock et al., 1996). Moreover, it has been shown that constructing a minimal binary tree with respect to the expected number of tests required for classifying an unseen instance is NP–complete (Hyafil and Rivest, 1976). Even finding the minimal equivalent decision tree for a given decision tree (Zantema and Bodlaender, 2000) or building the optimal decision tree from decision tables is known to be NP–hard (Naumov, 1991). The above results indicate that using optimal decision tree algorithms is feasible only in small problems. Consequently, heuristics methods are required for solving the problem. Roughly speaking, these methods can be divided into two groups: top–down and bottom–up with clear preference in the literature to the first group.

There are various top–down decision trees inducers such as ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), CART (Breiman et al., 1984). Some consist of two conceptual phases: growing and pruning (C4.5 and CART). Other inducers perform only the growing phase. Figure 9.2 presents a typical algorithmic framework for top–down inducing of a decision tree using growing and pruning. Note that these algorithms are greedy by nature and construct the decision tree in a top–down, recursive manner (also known as “divide and conquer“). In each iteration, the algorithm considers the partition of the training set using the outcome of a discrete function of the input attributes. The selection of the most appropriate function is made according to some splitting measures. After the selection of an appropriate split, each node further subdivides the training set into smaller subsets, until no split gains sufficient splitting measure or a stopping criterion is satisfied.

1. **METHOD**

BASED ON UNIVARIATE SPLITTING CRITERIA

Overview — In most of the cases, the discrete splitting functions are univariate. Univariate means that an internal node is split according to the value of a single attribute. Consequently, the inducer searches for the best attribute upon which to split. There are various univariate criteria. These criteria can be characterized in different ways, such as:

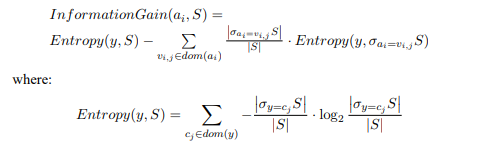
i) According to the origin of the measure: information theory, dependence, and distance.

ii) According to the measure structure: impurity-based criteria, normalized impurity-based criteria and Binary criteria.

The following section describes some of the most common criteria.

A) Information Gain

Information gain is an impurity-based criterion that uses the entropy measure (origin from information theory) as the impurity measure (Quinlan, 1987).



B) Gini Index

Gini index is an impurity-based criterion that measures the divergences between the probability distributions of the target attribute’s values. The Gini index has been used in various works such as (Breiman et al., 1984) and (Gelfand et al., 1991) and it is defined as:



Consequently, the evaluation criterion for selecting the attribute ai is defined as:



C) Gain Ratio

The gain ratio “normalizes” the information gain as follows (Quinlan, 1993):



Note that this ratio is not defined when the denominator is zero. Also the ratio may tend to favor attributes for which the denominator is very small. Consequently, it is suggested in two stages. First the information gain is calculated for all attributes. As a consequence, taking into consideration only attributes that have performed at least as good as the average information gain, the attribute that has obtained the best ratio gain is selected. It has been shown that the gain ratio tends to outperform simple information gain criteria, both from the accuracy aspect, as well as from classifier complexity aspects (Quinlan, 1988).

Comparison of Univariate Splitting Criteria

Comparative studies of the splitting criteria described above, and others, have been conducted by several researchers during the last thirty years, such as (Baker and Jain, 1976; BenBassat, 1978; Mingers, 1989; Fayyad and Irani, 1992; Buntine and Niblett, 1992; Loh and Shih, 1997; Loh and Shih, 1999; Lim et al., 2000). Most of these comparisons are based on empirical results, although there are some theoretical conclusions. Many of the researchers point out that in most of the cases, the choice of splitting criteria will not make much difference on the tree performance.

DECISION TREE INDUCERS - CART

CART stands for Classification and Regression Trees (Breiman et al., 1984). It is characterized by the fact that it constructs binary trees, namely each internal node has exactly two outgoing edges. The splits are selected using the twoing criteria and the obtained tree is pruned by cost–complexity Pruning. When provided, CART can consider misclassification costs in the tree induction. It also enables users to provide prior probability distribution. An important feature of CART is its ability to generate regression trees. Regression trees are trees where their leaves predict a real number and not a class. In case of regression, CART looks for splits that minimize the prediction squared error (the least–squared deviation). The prediction in each leaf is based on the weighted mean for node.

1. **ADVANTAGES AND DISADVANTAGES OF DECISION TREES**

Several advantages of the decision tree as a classification tool have been pointed out in the literature:

1. Decision trees are self–explanatory and when compacted they are also easy to follow. In other words, if the decision tree has a reasonable number of leaves, it can be grasped by non–professional users. Furthermore, decision trees can be converted to a set of rules. Thus, this representation is considered as comprehensible.

2. Decision trees can handle both nominal and numeric input attributes.

3. Decision tree representation is rich enough to represent any discrete– value classifier.

4. Decision trees are capable of handling datasets that may have errors.

5. Decision trees are capable of handling datasets that may have missing values.

6. Decision trees are considered to be a nonparametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

On the other hand, decision trees have such disadvantages as:

1. Most of the algorithms (like ID3 and C4.5) require that the target attribute will have only discrete values.

2. As decision trees use the “divide and conquer” method, they tend to perform well if a few highly relevant attributes exist, but less so if many complex interactions are present. One of the reasons for this is that other classifiers can compactly describe a classifier that would be very challenging to represent using a decision tree. A simple illustration of this phenomenon is the replication problem of decision trees (Pagallo and Huassler, 1990). Since most decision trees divide the instance space into mutually exclusive regions to represent a concept, in some cases the tree should contain several duplications of the same sub-tree in order to represent the classifier.

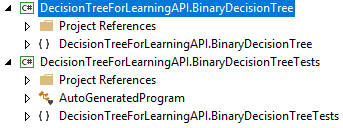
3. The greedy characteristic of decision trees leads to another disadvantage that should be pointed out. This is its over–sensitivity to the training set, to irrelevant attributes and to noise (Quinlan, 1993).

1. **PROJECT**

A) Git Hub reference

<https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/se-saisudhamurali/MyWork/DecisionTrees>

B) Solution overview



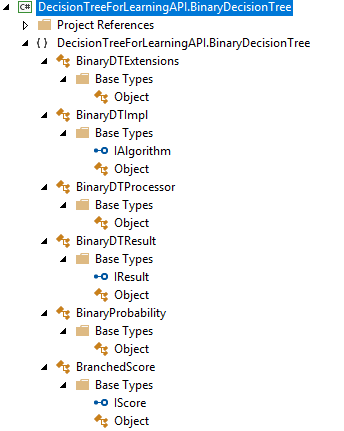
C) Implementation

1.BinaryDTImpl: Main class implementing the IAlgorithm from LearningAPI.

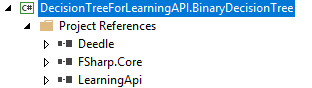
2.BinaryDTProcessor: Core class that processes data to be trained and predicted.

3.BranchedScore: Output of train/run method, implementing IScore from LearaningAPI.

4.BinaryDTResult: Output of predict method, implementing IResult from LearningAPI.

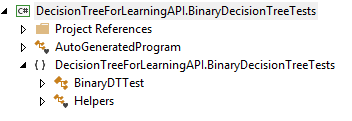


High level project references

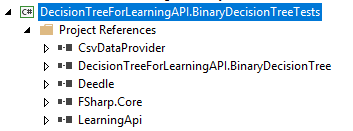


D) Testing

BinaryDTTest: Test class



High level project references



E**)** Limitations

Training and Testing datasets should contain numerical columns only.

**VII. REFERENCES**

[**https://scikit-learn.org/stable/supervised\_learning.html#supervised-learning**](https://scikit-learn.org/stable/supervised_learning.html#supervised-learning)

[**https://lethalbrains.com/learn-ml-algorithms-by-coding-decision-trees-439ac503c9a4**](https://lethalbrains.com/learn-ml-algorithms-by-coding-decision-trees-439ac503c9a4)

[**http://www.ise.bgu.ac.il/faculty/liorr/hbchap9.pdf**](http://www.ise.bgu.ac.il/faculty/liorr/hbchap9.pdf)

# Bibliography

**There are no sources in the current document.**