DTrules.py Documentation

author: Samir Farooq
company: University of Rochester Medical Center
team: Rochester Center for Health Informatics
email: samir_farooq@urmc.rochester.edu or martin_zand@urmc.rochester.edu
last revised: 12 June 2018

The use of this file is to extract rules from a decision tree created by the sci-kit learn implementation in python. This program requires the packages: numpy and matplotlib. This file includes four different types of classes: Node, Queue, Tree_Node, and DTrule_extraction. The only class which the user should concern himself with is DTrule_extraction, as all other classes and functions in the file are used as back-end methods for this class.

class `DTrules.DTrule_extraction (DT):`

This class has been designed to extract rules from a sci-kit learn implementation of the decision tree algorithm in python.

Parameters:

`DT` : Decision Tree object from sci-kit learn

Decision Tree object from sci-kit learn to extract rules from. The decision tree must already be fitted to the model.

Attributes:

`unclean_rulebook` : dictionary

This data structure holds the extracted rules. The keys of the dictionary are the classes and the values are the rules: unclean_rulebook = {class_1: rules_1, class_2: rules_2, class_3: rules_3, ..., class_n: rules_n}. Each i'th rules are in the format of a list: rules_i = [rule_1, rule_2, rule_3, ..., rule_m]. Each i'th rule itself is a list: rule_i = [feature interval_1, feature interval_2, feature interval_3, ..., feature interval_k, support, purity, depth]. Notice that the last three elements of this list are unrelated to the feature rules, rather they give information on the rule's support, purity, and depth. Support = $\frac{true\ positive}{all\ positive}$, Purity = $\frac{true\ positive+false\ positive}{true\ positive}$, and depth is an integer equivalent to how deep the rule goes into the decision tree (in terms of nodes). Each i'th feature interval is a list composed of two values, the minimum value and maximum value: feature interval_i = [minimum, maximum]. If minimum = None then the minimum does not exist (i.e. $-\infty$), and if maximum = None then the maximum does not exist (i.e. ∞). It is possible for both minimum and maximum to be equal to None, which means that the rule is independent of the value of that feature.

`DT` : Decision Tree object from sci-kit learn

The inputted Decision Tree object that rules are extracted from (God willing).

`max_depth` : integer

The maximum depth that the decision tree takes.

`head_node` : object of the class `DTrules.Tree_Node`

The sci-kit learn's implementation of a decision is converted into a tree-like data structure rather than numpy arrays of node existence, which made rule extraction easier. This particular attribute holds the head of the tree-structure.

`leaves` : object of the class `DTrules.Queue`

This object is used as part of generating the rules leaf-up. This queue should be empty because the generation of rules is automatically called. Hence this attribute should not concern the end-user.

Methods:

plot (feat_names=None, minimum=0, maximum=1, class_colors={}, c2f=0.05, f2b=0.01, b2b=0.15, b2c=0.05, b_size=0.05, b2s_ratio=0.9, b2u=0.005, c_size=18, f_size=10, u_size=10, s_size=12, dr2cr=0.075, title=''Decision Tree Rules'', Centroid=None, save='Fig7.pdf')

Plots the decision rules that were generated. feat_names should be a list or numpy array whose elements are the feature names corresponding to the column number that the decision tree was fit to. If feat_names=None, then dummy variables will be used in their stead. minimum is the minimum value that the features can take (the input can be an integer, float, or an array if each feature has different minimums. maximum is the maximum value that the features can take (input type is the same as minimum). Currently this function does not support features who have no minimum or have no maximum (rather it assumes that the features have been normalized to some sort of strict domain). class_colors is a dictionary whose keys are the classes and whose values correspond to some rgb color code or a color string code recognized by matplotlib. c2f controls the spacing between the class name and the feature names. f2b controls the spacing between the feature name and the feature bars. b2b controls the spacing between feature bars. b2c controls the spacing between the last feature bar and the next class name. b_size controls the vertical size of the feature bar. b2s_ratio controls the horizontal size of the feature bar which should be a value between [0,1] (where 1 would mean there is no horizontal spacing between feature bars). b2u controls the spacing between the feature bar and the feature units. c_size is the fontsize of the class names and the title. f_size is the fontsize of the feature names. u_size is the fontsize of the feature units. s_size is the fontsize of the support, purity, and depth statistics. dr2cr controls the spacing between the decision rules and the centroid rules (when applicable). title is the title of the plot, whose fontsize is also controlled by c_size. If the user wishes the compare the decision rules with Centroid rules then they can input a fitted Centroid into the Centroid parameter. save controls what name to save the plot with (input False if saving is not desired).

unclean2cleanrule (rule, feat_names=None)

Takes a rule and gives a clean representation of that rule using the provided feature names (dummy variables corresponding to the column of the feature if None).

set_up_tree_structure ()

Back-end method which converts the decision tree into a tree-like structure.

update_class_choices ()

Back-end method which keeps track of the purity of rules in order to avoid generating rules which are too specific.

generate_rules ()

Back-end method which generates the rules using a leaf-up method (i.e. starting at each leaf node and then moving up the tree to determine the rule).

rule_of_node (node)

Back-end method of the generate_rules method which extracts the rule of a leaf node.