# **Table of contents**

[Table of contents 1](#__RefHeading___Toc9571_2303100741)

[1. Introduction 1](#__RefHeading___Toc9573_2303100741)

[1.1. Algorithm inputs 1](#__RefHeading___Toc9575_2303100741)

[1.2. Algorithm output 2](#__RefHeading___Toc9577_2303100741)

# Introduction

iSOUP-Trees are decision trees that are learned incrementally and can be used for modeling of structured target variables. Decision trees in general, are hierarchical models, where each internal node contains a test on a descriptive variable (attribute) of an example and each branch leaving this node corresponds to an outcome of this test. Terminal nodes (leaves) of a tree contain models defining the values of the target (dependent) variable for all examples falling in a given leaf. Given a new example, for which the value of the target variable should be predicted, the tree is interpreted from the root. In each inner node, the prescribed test is performed, and according to the result, the corresponding sub-tree is selected. When the selected node is a leaf, the value of the target variable for the new example is predicted according to the model in this leaf. In iSOUP-Tree regression trees the model in a leaf is a constant model. iSOUP-Trees process the target variable as a structured object, in particular, a vector of numeric values. These trees thus predict several target variables simultaneously. The main advantage of such trees over a set of separate trees for each dependent variable is that a single tree is usually much smaller than the total size of the individual trees for all variables and therefore much easier to interpret. Once a decision tree is learned on the data we can use it for two purposes. First, we can use it for explaining the connections between the variables and determine the variables that are important for grouping (clustering) similar examples according to the dependent target variable. Second, we can use the tree for predicting the target variable of new examples.

Due to the nature of their learning procedure, iSOUP-Trees process data examples one by one and never store the examples directly. This allows us to transfer models between hospital nodes without violating privacy restrictions. Once a model is transferred to a new hospital node it is further updated using the available data. The process then repeats for the next hospital node until the model has learned from all of the available data. This allows the model to generalize over all available data.

The algorithm takes the following parameters:

* *Target variables*: A set of one or more (dependent) variables for which we want to learn a predictive model that will describe their values in terms of descriptive variables. They must all be numeric.

## Algorithm inputs

This algorithm supports multi-target regression, i.e., all target attributes must be numeric.

## Algorithm output

The algorithm produces two outputs: a PFA action and a VisJS visualization. Internally, it produces a third output which is the serialized model that is transferred between hospital nodes. Below we provide output for different output types. The PFA action is only part of the PFA file, as the algorithm does not have direct access to the system variables. This is later fully annotated within the Exareme framework.

#### PFA action

|  |
| --- |
| {  "doc" : "ISOUP-RegressionTree",  "action" : [ {  "if" : {  "<=" : [ "input.hr", 6.0 ]  },  "then" : {  "if" : {  "<=" : [ "input.hr", 0.0 ]  },  "then" : {  "if" : {  "<=" : [ "input.workingday", 0.0 ]  },  "then" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  },  "else" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  }  },  "else" : {  "if" : {  "<=" : [ "input.workingday", 0.0 ]  },  "then" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  },  "else" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  }  }  },  "else" : {  "if" : {  "<=" : [ "input.hr", 0.0 ]  },  "then" : {  "if" : {  "<=" : [ "input.workingday", 0.0 ]  },  "then" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  },  "else" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  }  },  "else" : {  "if" : {  "<=" : [ "input.workingday", 0.0 ]  },  "then" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  },  "else" : {  "casual" : 17.583333333333332,  "registered" : 77.38425925925925,  "cnt" : 94.9675925925926  }  }  }  } ]  } |

#### VISjs

|  |
| --- |
| var nodes=[]; var edges=[];  nodes.push({id: 0, label: 'hr <= 6.0'});  nodes.push({id: 1, label: 'hr <= 0.0'});  nodes.push({id: 2, label: 'workingday <= 0.0'});  nodes.push({id: 3, label: '[17.583333333333332; 77.38425925925925; 94.9675925925926]'});  nodes.push({id: 4, label: '[7.11377245508982; 29.79241516966068; 36.9061876247505]'});  nodes.push({id: 5, label: 'hr <= 6.0'});  nodes.push({id: 6, label: 'hr <= 5.0'});  nodes.push({id: 7, label: 'workingday <= 0.0'});  nodes.push({id: 8, label: 'hr <= 2.0'});  nodes.push({id: 9, label: 'season <= 1.0'});  nodes.push({id: 10, label: '[4.449275362318841; 35.73913043478261; 40.18840579710145]'});  nodes.push({id: 11, label: '[14.602941176470589; 61.16544117647059; 75.76838235294117]'});  nodes.push({id: 12, label: '[4.496515679442509; 13.95993031358885; 18.45644599303136]'});  nodes.push({id: 13, label: 'hr <= 5.0'});  nodes.push({id: 14, label: 'hr <= 4.0'});  nodes.push({id: 15, label: 'hr <= 1.0'});  nodes.push({id: 16, label: '[3.59375; 14.70138888888889; 18.29513888888889]'});  nodes.push({id: 17, label: 'hr <= 2.0'});  nodes.push({id: 18, label: '[2.327956989247312; 8.505376344086022; 10.833333333333334]'});  nodes.push({id: 19, label: '[1.0278450363196125; 5.190072639225182; 6.217917675544794]'});  nodes.push({id: 20, label: 'temp <= 0.72'});  nodes.push({id: 21, label: '[1.7151898734177216; 33.462025316455694; 35.177215189873415]'});  nodes.push({id: 22, label: '[1.1650943396226414; 22.84433962264151; 24.00943396226415]'});  nodes.push({id: 23, label: '[2.541772151898734; 8.832911392405064; 11.374683544303798]'});  nodes.push({id: 24, label: 'weekday <= 5.0'});  nodes.push({id: 25, label: '[4.571428571428571; 91.4748490945674; 96.04627766599597]'});  nodes.push({id: 26, label: '[3.9283489096573208; 24.186915887850468; 28.115264797507788]'});  nodes.push({id: 27, label: '[1.4362139917695473; 7.495884773662551; 8.932098765432098]'});  nodes.push({id: 28, label: 'hr <= 18.0'});  nodes.push({id: 29, label: 'hr <= 18.0'});  nodes.push({id: 30, label: 'weathersit <= 1.0'});  nodes.push({id: 31, label: 'hr <= 18.0'});  nodes.push({id: 32, label: 'workingday <= 0.0'});  nodes.push({id: 33, label: 'hr <= 9.0'});  nodes.push({id: 34, label: '[28.594444444444445; 99.00833333333334; 127.60277777777777]'});  nodes.push({id: 35, label: 'temp <= 0.34'});  nodes.push({id: 36, label: '[35.586021505376344; 156.99462365591398; 192.58064516129033]'});  nodes.push({id: 37, label: '[153.78065134099617; 260.7911877394636; 414.57183908045977]'});  nodes.push({id: 38, label: 'hr <= 15.0'});  nodes.push({id: 39, label: 'hr <= 8.0'});  nodes.push({id: 40, label: 'hr <= 7.0'});  nodes.push({id: 41, label: '[12.625698324022347; 349.340782122905; 361.9664804469274]'});  nodes.push({id: 42, label: '[22.844036697247706; 520.7752293577981; 543.6192660550458]'});  nodes.push({id: 43, label: 'hr <= 9.0'});  nodes.push({id: 44, label: '[28.38364779874214; 268.8930817610063; 297.27672955974845]'});  nodes.push({id: 45, label: 'mnth <= 2.0'});  nodes.push({id: 46, label: '[17.63157894736842; 133.28571428571428; 150.91729323308272]'});  nodes.push({id: 47, label: '[54.779041429731926; 172.08529650690497; 226.86433793663687]'});  nodes.push({id: 48, label: '[60.68641114982579; 394.4634146341463; 455.1498257839721]'});  nodes.push({id: 49, label: '[62.44736842105263; 179.4440789473684; 241.89144736842104]'});  nodes.push({id: 50, label: 'atemp <= 0.7273'});  nodes.push({id: 51, label: 'hr <= 8.0'});  nodes.push({id: 52, label: '[18.410810810810812; 390.9837837837838; 409.3945945945946]'});  nodes.push({id: 53, label: 'mnth <= 10.0'});  nodes.push({id: 54, label: '[80.10211267605634; 289.3943661971831; 369.4964788732394]'});  nodes.push({id: 55, label: '[56.39688359303391; 214.67277726856096; 271.06966086159485]'});  nodes.push({id: 56, label: '[28.579636363636364; 157.46836363636365; 186.048]'});  nodes.push({id: 57, label: '[18.538699690402478; 84.36532507739938; 102.90402476780186]'});  nodes.push({id: 58, label: 'hr <= 19.0'});  nodes.push({id: 59, label: 'mnth <= 10.0'});  nodes.push({id: 60, label: '[66.67078189300412; 304.37654320987656; 371.0473251028807]'});  nodes.push({id: 61, label: '[13.608108108108109; 188.11711711711712; 201.72522522522522]'});  nodes.push({id: 62, label: 'hr <= 21.0'});  nodes.push({id: 63, label: 'season <= 3.0'});  nodes.push({id: 64, label: '[50.77; 207.65; 258.42]'});  nodes.push({id: 65, label: '[17.209119496855347; 142.92767295597486; 160.13679245283018]'});  nodes.push({id: 66, label: '[16.313791178112787; 86.3964265773311; 102.71021775544389]'});  edges.push({from: 2, to: 3, label: 'No'});  edges.push({from: 2, to: 4, label: 'Yes'});  edges.push({from: 1, to: 2, label: 'No'});  edges.push({from: 9, to: 10, label: 'No'});  edges.push({from: 9, to: 11, label: 'Yes'});  edges.push({from: 8, to: 9, label: 'No'});  edges.push({from: 8, to: 12, label: 'Yes'});  edges.push({from: 7, to: 8, label: 'No'});  edges.push({from: 15, to: 16, label: 'No'});  edges.push({from: 17, to: 18, label: 'No'});  edges.push({from: 17, to: 19, label: 'Yes'});  edges.push({from: 15, to: 17, label: 'Yes'});  edges.push({from: 14, to: 15, label: 'No'});  edges.push({from: 20, to: 21, label: 'No'});  edges.push({from: 20, to: 22, label: 'Yes'});  edges.push({from: 14, to: 20, label: 'Yes'});  edges.push({from: 13, to: 14, label: 'No'});  edges.push({from: 13, to: 23, label: 'Yes'});  edges.push({from: 7, to: 13, label: 'Yes'});  edges.push({from: 6, to: 7, label: 'No'});  edges.push({from: 24, to: 25, label: 'No'});  edges.push({from: 24, to: 26, label: 'Yes'});  edges.push({from: 6, to: 24, label: 'Yes'});  edges.push({from: 5, to: 6, label: 'No'});  edges.push({from: 5, to: 27, label: 'Yes'});  edges.push({from: 1, to: 5, label: 'Yes'});  edges.push({from: 0, to: 1, label: 'No'});  edges.push({from: 33, to: 34, label: 'No'});  edges.push({from: 35, to: 36, label: 'No'});  edges.push({from: 35, to: 37, label: 'Yes'});  edges.push({from: 33, to: 35, label: 'Yes'});  edges.push({from: 32, to: 33, label: 'No'});  edges.push({from: 40, to: 41, label: 'No'});  edges.push({from: 40, to: 42, label: 'Yes'});  edges.push({from: 39, to: 40, label: 'No'});  edges.push({from: 43, to: 44, label: 'No'});  edges.push({from: 45, to: 46, label: 'No'});  edges.push({from: 45, to: 47, label: 'Yes'});  edges.push({from: 43, to: 45, label: 'Yes'});  edges.push({from: 39, to: 43, label: 'Yes'});  edges.push({from: 38, to: 39, label: 'No'});  edges.push({from: 38, to: 48, label: 'Yes'});  edges.push({from: 32, to: 38, label: 'Yes'});  edges.push({from: 31, to: 32, label: 'No'});  edges.push({from: 31, to: 49, label: 'Yes'});  edges.push({from: 30, to: 31, label: 'No'});  edges.push({from: 51, to: 52, label: 'No'});  edges.push({from: 53, to: 54, label: 'No'});  edges.push({from: 53, to: 55, label: 'Yes'});  edges.push({from: 51, to: 53, label: 'Yes'});  edges.push({from: 50, to: 51, label: 'No'});  edges.push({from: 50, to: 56, label: 'Yes'});  edges.push({from: 30, to: 50, label: 'Yes'});  edges.push({from: 29, to: 30, label: 'No'});  edges.push({from: 29, to: 57, label: 'Yes'});  edges.push({from: 28, to: 29, label: 'No'});  edges.push({from: 59, to: 60, label: 'No'});  edges.push({from: 59, to: 61, label: 'Yes'});  edges.push({from: 58, to: 59, label: 'No'});  edges.push({from: 63, to: 64, label: 'No'});  edges.push({from: 63, to: 65, label: 'Yes'});  edges.push({from: 62, to: 63, label: 'No'});  edges.push({from: 62, to: 66, label: 'Yes'});  edges.push({from: 58, to: 62, label: 'Yes'});  edges.push({from: 28, to: 58, label: 'Yes'});  edges.push({from: 0, to: 28, label: 'Yes'});  var container = document.getElementById('visualization');var data = { nodes: nodes, edges: edges};var options = { layout: { hierarchical: { sortMethod: layoutMethod } }, edges: { smooth: true, arrows: {to : true} }};network = new vis.Network(container, data, options); |