**Decision Tree Regression**

Decision tree builds regression or classification models in the form of a tree structure. It brakes down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy), each representing values for the attribute tested. Leaf node (e.g., PlayTennis) represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

### **Overfitting**

Overfitting is a practical problem while building a decision tree model. The model is having an issue of overfitting is considered when the algorithm continues to go deeper and deeper to reduce the training set error but results with an increased test set error i.e, Accuracy of prediction for our model goes down. It generally happens when it builds many branches due to outliers and irregularities in data.

Two approaches which we can use to avoid overfitting are:

* Pre-Pruning
* Post-Pruning

### **Pre-Pruning**

In pre-pruning, it stops the tree construction bit early. It is preferred not to split a node if its goodness measure is below a threshold value. But it’s difficult to choose an appropriate stopping point.

## **Post-Pruning**

In post-pruning first, it goes deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem then pruning is done as a post-pruning step. We use a cross-validation data to check the effect of our pruning. Using cross-validation data, it tests whether expanding a node will make an improvement or not.

If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded i.e, the node should be converted to a leaf node.

## Decision Tree Algorithm Advantages and Disadvantages

## Advantages:

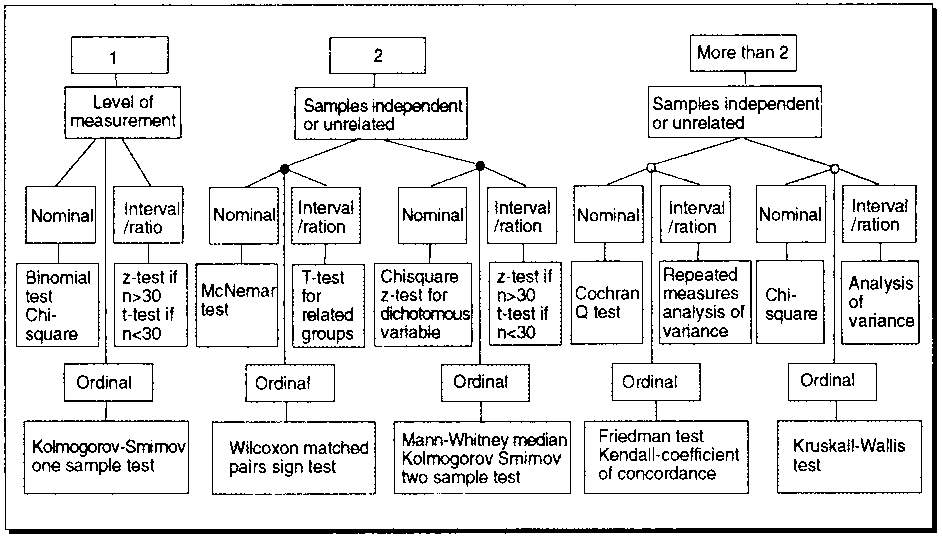
1. Decision Trees are easy to explain. It results in a set of rules.
2. It follows the same approach as humans generally follow while making decisions.
3. Interpretation of a complex Decision Tree model can be simplified by its visualizations. Even a naive person can understand logic.
4. The Number of hyper-parameters to be tuned is almost null.

## Disadvantages:

1. There is a high probability of overfitting in Decision Tree.
2. Generally, it gives low prediction accuracy for a dataset as compared to other machine learning algorithms.
3. Information gain in a decision tree with categorical variables gives a biased response for attributes with greater no. of categories.
4. Calculations can become complex when there are many class labels.

**Decode Complex Algorithm**:

1. IN GRAPH :



(ii) IN Calculation

L: \textup{Left child node for the root node} 

R: \textup{Right child node for the root node} 

P_{L}= \frac{\textup{Number of records in left child node}}{\textup{Total number of records}}  

P_{R}= \frac{\textup{Number of records in right child node}}{\textup{Total number of records}}  

P(k|\textup{L})=\frac{\textup{Number of class k records in left child node}}{\textup{Number of records in left child node}}  

P(k|\textup{R})=\frac{\textup{Number of class k records in right child node}}{\textup{Number of records in right child node}}  

\textup{Goodness of Split}=2P_{L}P_{R}\times \sum_{k=0,1}\left | P(k|\textup{L})-P(k|\textup{R}) \right | 

\therefore \textup{Goodness of Split}=\Psi (\textup{Large Piece})\times \Psi (\textup{Pick Cherries}) 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Activity in the Last Quarter** | **Number of Solicited Customers** | **Campaign Results** |  | **Success Rate** |
|  |  | Responded (***r*)** | Not Responded (***nr*)** |  |
| low | 40000 | 720 | 39280 | 0.018 |
| medium | 30000 | 1380 | 28620 | 0.046 |
| high | 30000 | 2100 | 27900 | 0.07 |
| Total | 100000 |  |  |  |

P_{L}=\frac{\textup{\# customers in Low}}{\textup{All the customers }}=\frac{40000}{100000}=0.4  

P_{R}=\frac{\textup{\# customers in Medium + High}}{\textup{All the customers }}=\frac{60000}{100000}=0.6  

Now calculating  Ψ(Large Piece) as below:

|  |
| --- |
| \Psi (\textup{Large Piece})=2P_{L}P_{R}=2\times0.4\times 0.6=0.48 |

Now, let’s come to the second part of the equation that is Ψ(Pick Cherries).

r = responded and

nr = not-responded customers for campaign’s .

|  |
| --- |
| \textup{r: }P(k|L)=\frac{\textup{\# customers responded in Low}}{\textup{Total number of customers in Low}}=\frac{720}{40000}=0.018 |

|  |
| --- |
| r : P(k|(M+H)) =#Customer responded in (Medium+High)/Total number of customers in (Medium+High) =(1380+2100)/(30000+30000)=3480/60000 |
| =0.05 |

|  |
| --- |
| \textup{nr: } P(k|L)=\frac{\textup{\# customers not responded in Low}}{\textup{Total number of customers in Low}}=\frac{39280}{40000}=0.982 |

|  |
| --- |
| nr : P(k|(M+H)) =#Customer not responded in (Medium+High)/Total number of customers in (Medium+High) =(28620+27900)/(30000+30000)=56520/60000 |
| =0.942 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Left Node** | **Right Node** | **P(Left)** | **P(Right)** | **P(k|L)  = a** | **P(k|R)  = b** | **Ψ(Large Piece)** | **Ψ(Pick Cherries)** | **Goodness of Split** |
|  |  |  |  |  |  | 2P**LPR** | Σ(a-b) |  |
| Low | Medium+High | 0.4 | 0.6 | r: 0.018 | r: 0.058 | 0.48 | 0.08 | 0.0384 |
|  |  |  |  | nr: 0.982 | nr: 0.942 |  |  |  |
| Low+Medium | High | 0.7 | 0.3 | r: 0.030 | r: 0.070 | 0.42 | 0.08 | 0.0336 |
|  |  |  |  | nr: 0.970 | nr: 0.930 |  |  |  |
| Low+high | Medium | 0.7 | 0.3 | r: 0.040 | r: 0.046 | 0.42 | 0.011 | 0.0048 |

**Calculation:**

Calculating r: P(k|R), and nr: P(k|R))

\Psi(\textup{Pick Cherries})=\left |P(r|L)-P(r|R) \right |+\left | P(nr|L)-P(nr|R) \right| 

This leaves us with one last calculation for the last column i.e.  goodness of split which is:

\textup{Goodness of split}=\Psi(\textup{Large Piece})\times \Psi (\textup{Pick Cherries})=0.48\times 0.080 

\therefore \Psi(\textup{Pick Cherries})=|0.018-0.058|+|0.982-0.942|=0.080 

\therefore \textup{Goodness of split} =0.0384 



**Simple to understand and interpret.**

**Able to handle both numerical and** [**categorical**](https://en.wikipedia.org/wiki/Categorical_variable) **data**

**Requires little data preparation**

**Possible to validate a model using statistical tests**

**Performs well with large datasets.**

USE CASES

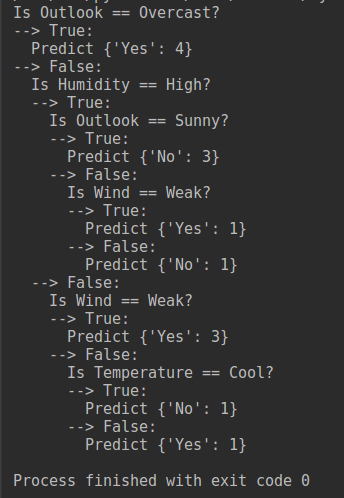
* **Agriculture:** Application of a range of machine learning methods to problems in agriculture and horticulture is described in.
* **Astronomy:** Astronomy has been an active domain for using automated classification techniques. Use of decision trees for filtering noise from Hubble Space Telescope images was reported recently in. Decision trees have helped in star-galaxy classification, determining galaxy counts and discovering quasars in the Second Palomar Sky Survey. Use of neural trees for ultraviolet stellar spectral classification is described in
* **Biomedical Engineering:** Use of decision trees for identifying features to be used in implantable devices can be found in
* **Control Systems:** Automatic induction of decision trees was recently used for control of nonlinear dynamical systems
* **Financial analysis:** Use of CART for asserting the attractiveness of buy-writes is reported in.
* **Medicine:** Medical research and practice have long been important areas of application for decision tree techniques. Recent uses of automatic induction of decision trees can be found in diagnosis , cardiology, psychiatry , gastroenterology , for detecting microcalcifications in mammography , to analyze Sudden Infant Death (SID) syndrome and for diagnosing thyroid disorders .
* **Molecular biology:** Initiatives such as the Human Genome Project and the GenBank database offer fascinating opportunities for machine learning and other data exploration methods in molecular biology.
* **Object recognition:** Tree based classification has been used recently for recognizing three dimensional objects and for high level vision .
* **Pharmacology:** Use of tree-based classification for drug analysis can be found.
* **Physics:** Decision trees have been used for the detection of physical particles.
* **Plant diseases:** CART was recently used to assess the hazard of mortality to pine trees.
* **Power systems:** Power system security assessment and power stability prediction are two areas in power systems maintenance for which decision trees were used.
* **Remote Sensing:** Remote sensing has been a strong application area for pattern recognition work on decision trees. A recent use of tree-based classification in remote sensing can be found.
* **Software development:** Regression trees (and backpropagation networks) were recently used to estimate the development effort of a given software module in, where it is argued that machine learning methods compare favourably with traditional methods.

**-Banking Sector to facilitate loan approving criterion.**

**-In pharmacy to determine which medine is proper for a particular cure.**

**-In automotive / F1 to determine which tyre to use on certain track in a particular weather.**

1. Finding the difference between approval of loan among Emp maximum and minimum salary.
2. Finding obese person with adiposity prone to heart disease
3. Finding Product Price & Sales
4. Finding Age & Mortality
5. Finding temperature vs. Number of cones sold at ice cream store
6. Finding Population vs Food consumption
7. Finding quantity with yield
8. Determining the chances to win cricket match .
9. Determining the chances of getting Jobs after Completing Graduation.
10. Speed and distance relationship
11. Finding rate of growth of the economy of a Institution
12. PYTHON CODE
13. Decision Tree Code
14. from \_\_future\_\_ import print\_function
15. import matplotlib.pyplot as plt
16. import numpy as np
17. import pandas as pd
18. #
19. # training\_data = [
20. # ['Green', 3, 'Apple'],
21. # ['Yellow', 3, 'Apple'],
22. # ['Red', 1, 'Grape'],
23. # ['Red', 1, 'Grape'],
24. # ['Yellow', 3, 'Lemon'],
25. # ]
26. #header = ["color", "diameter", "label"]#used to print the tree
27. training\_data = [
28. ['Sunny','Hot','High','Weak','No'],
29. ['Sunny','Hot','High','Strong','No'],
30. ['Overcast','Hot','High','Weak','Yes'],
31. ['Rain', 'Mild', 'High', 'Weak', 'Yes'],
32. ['Rain','Cool','Normal','Weak','Yes'],
33. ['Rain','Cool','Normal','Strong','No'],
34. ['Overcast','Cool','Normal','Strong','Yes'],
35. ['Sunny','Mild','High','Weak','No'],
36. ['Sunny','Cool','Normal','Weak','Yes'],
37. ['Rain','Mild','Normal','Weak','Yes'],
38. ['Sunny','Mild','Normal','Strong','Yes'],
39. ['Overcast','Mild','High','Strong','Yes'],
40. ['Overcast','Hot','Normal','Weak','Yes'],
41. ['Rain','Mild','High','Strong','No'],
42. ]
43. header=["Outlook","Temperature","Humidity","Wind","PlayTennis"]
44. def unique\_vals(rows, col):
45. return set([row[col] for row in rows]) #Find the unique values for a column in a dataset.
46. #print(unique\_vals(training\_data,0))
47. def class\_counts(rows):
48. #"""Counts the number of each type of example in a dataset."""
49. counts = {} # a dictionary of label -> count.
50. for row in rows:
51. # in our dataset format, the label is always the last column
52. label = row[-1]
53. if label not in counts:
54. counts[label] = 0
55. counts[label] += 1
56. return counts
57. #print(class\_counts(training\_data))
58. def is\_numeric(value):
59. return isinstance(value, int) or isinstance(value, float)
60. #print(is\_numeric("Red"))
61. class Question:
62. def \_\_init\_\_(self, column, value):
63. self.column = column
64. self.value = value
65. def match(self, example):
66. # Compare the feature value in an example to the
67. # feature value in this question.
68. val = example[self.column]
69. if is\_numeric(val):
70. return val >= self.value
71. else:
72. return val == self.value
73. def \_\_repr\_\_(self):
74. # This is just a helper method to print
75. # the question in a readable format.
76. condition = "=="
77. if is\_numeric(self.value):
78. condition = ">="
79. return "Is %s %s %s?" % (header[self.column], condition, str(self.value))
80. def partition(rows, question):
81. true\_rows, false\_rows = [], []
82. for row in rows:
83. if question.match(row):
84. true\_rows.append(row)
85. else:
86. false\_rows.append(row)
87. return true\_rows, false\_rows
88. def gini(rows):
89. counts = class\_counts(rows)
90. impurity = 1
91. for label in counts:
92. prob\_of\_label = counts[label] / float(len(rows))
93. impurity -= prob\_of\_label\*\*2
94. return impurity
95. def info\_gain(left, right, current\_uncertainty):
96. # """Information Gain.
97. # The uncertainty of the starting node, minus the weighted impurity of
98. # two child nodes.
99. # """
100. p = float(len(left)) / (len(left) + len(right))
101. return current\_uncertainty - p \* gini(left) - (1 - p) \* gini(right)
102. def find\_best\_split(rows):
103. # """Find the best question to ask by iterating over every feature / value
104. # and calculating the information gain."""
105. best\_gain = 0 # keep track of the best information gain
106. best\_question = None # keep train of the feature / value that produced it
107. current\_uncertainty = gini(rows)
108. n\_features = len(rows[0]) - 1 # number of columns
109. for col in range(n\_features): # for each feature
110. values = set([row[col] for row in rows]) # unique values in the column
111. for val in values: # for each value
112. question = Question(col, val)
113. # try splitting the dataset
114. true\_rows, false\_rows = partition(rows, question)
115. # Skip this split if it doesn't divide the
116. # dataset.
117. if len(true\_rows) == 0 or len(false\_rows) == 0:
118. continue
119. # Calculate the information gain from this split
120. gain = info\_gain(true\_rows, false\_rows, current\_uncertainty)
121. # You actually can use '>' instead of '>=' here
122. # but I wanted the tree to look a certain way for our
123. # toy dataset.
124. if gain >= best\_gain:
125. best\_gain, best\_question = gain, question
126. return best\_gain, best\_question
127. class Leaf:
128. # """A Leaf node classifies data.
129. # This holds a dictionary of class (e.g., "Apple") -> number of times
130. # it appears in the rows from the training data that reach this leaf.
131. # """
132. def \_\_init\_\_(self, rows):
133. self.predictions = class\_counts(rows)
134. class Decision\_Node:
135. # """A Decision Node asks a question.
136. # This holds a reference to the question, and to the two child nodes.
137. # """
138. def \_\_init\_\_(self,
139. question,
140. true\_branch,
141. false\_branch):
142. self.question = question
143. self.true\_branch = true\_branch
144. self.false\_branch = false\_branch
145. def build\_tree(rows):
146. """Builds the tree.
147. Rules of recursion: 1) Believe that it works. 2) Start by checking
148. for the base case (no further information gain). 3) Prepare for
149. giant stack traces.
150. """
151. # Try partitioing the dataset on each of the unique attribute,
152. # calculate the information gain,
153. # and return the question that produces the highest gain.
154. gain, question = find\_best\_split(rows)
155. # Base case: no further info gain
156. # Since we can ask no further questions,
157. # we'll return a leaf.
158. if gain == 0:
159. return Leaf(rows)
160. # If we reach here, we have found a useful feature / value
161. # to partition on.
162. true\_rows, false\_rows = partition(rows, question)
163. # Recursively build the true branch.
164. true\_branch = build\_tree(true\_rows)
165. # Recursively build the false branch.
166. false\_branch = build\_tree(false\_rows)
167. # Return a Question node.
168. # This records the best feature / value to ask at this point,
169. # as well as the branches to follow
170. # dependingo on the answer.
171. return Decision\_Node(question, true\_branch, false\_branch)
172. def print\_tree(node, spacing=""):
173. """World's most elegant tree printing function."""
174. # Base case: we've reached a leaf
175. if isinstance(node, Leaf):
176. print (spacing + "Predict", node.predictions)
177. return
178. # Print the question at this node
179. print (spacing + str(node.question))
180. # Call this function recursively on the true branch
181. print (spacing + '--> True:')
182. print\_tree(node.true\_branch, spacing + " ")
183. # Call this function recursively on the false branch
184. print (spacing + '--> False:')
185. print\_tree(node.false\_branch, spacing + " ")
186. mytree=build\_tree(training\_data)
187. print\_tree(mytree)
188. # # plt.plot(training\_data[0],training\_data[2],'r+',training\_data[1],training\_data[2],'g-') #Plot y-vs-x in dots
189. # plt.plot(np.array(training\_data[0]).reshape(3,),np.array(mytree).reshape(3,1),'r+')
190. # plt.show()
191. **Output**



PYTHON CODE WITHOUT LIBRARY

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Position\_Salaries.csv')

X = dataset.iloc[:, 1:2].values

y = dataset.iloc[:, 2].values

from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor(random\_state = 0)

regressor.fit(X, y)

# Predicting a new result

y\_pred = regressor.predict(6.5)

# Visualising the Decision Tree Regression results (higher resolution)

X\_grid = np.arange(min(X), max(X), 0.01)

X\_grid = X\_grid.reshape((len(X\_grid), 1))

plt.scatter(X, y, color = 'red')

plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

plt.title('Truth or Bluff (Decision Tree Regression)')

plt.xlabel('Position level')

plt.ylabel('Salary')

plt.show()