Chapter 2 : Section 5

**Decision Tree Regression**

**2.5.1 Decision Tree**

* CART (Classification and Regression trees): Decision Trees are divided into *Classification* and *Regression* Trees.
* *Regression* *trees* are needed when the *response* *variable* is *numeric* or *continuous*.
* *Classification* *trees*, as the name implies are used to *separate* the *dataset* into classes belonging to the *response* *variable*.
* In this section, we discuss about: Decision Tree Regression Model

|  |  |
| --- | --- |
| * Assume that we've got a scatterplot which represent our Dataset. Here we've got two independent variables and . And we're *predicting* as the *third* *variable* which is *dependent* *variable*. It is in the 3rd axis and cannot be seen in 2-d. * We don't actually need to see "**y**" because we need to work with this scatterplot first, to build our decision tree. And then once we've built it will return to **y**. * In plane all points are the projection of the data points. |  |

* We need to work with this scatterplot to see how our decision tree is going to be created.
* So once you run the *Regression* *Tree* or *Decision* *Tree* *Algorithm* in Regression Sense of it, what will happen is your ***scatterplot*** will be Split Up into Segments. The spitted segments are called leaves. Let's have a look at how an algorithm do that:

|  |  |
| --- | --- |
| * Assume that the algorithm would create a split at somewhere around 20 (i.e. your diagram or your *scatterplot* now divided into two parts).  1. Everything and everything else that's . 2. For , then another split for is: and . 3. For , then other split for is and . 4. Lets again assume for and for there is a split for , which is and  * The Scatterplot is given in the Right: |  |

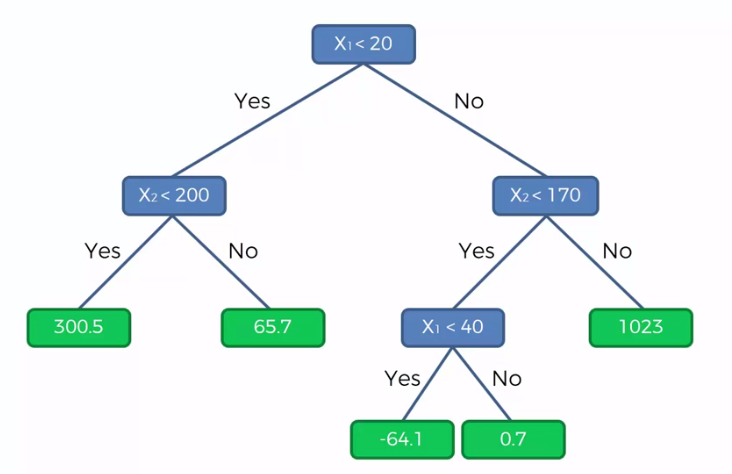
* How and where these splits are conducted is determined by the algorithm. And it is related to the ***Information*** ***Entropy***. When we perform split into our data (create a leaf), the *amount of information increases* (it actually adding some value when we group our points). The *algorithm* is finding the *optimal* *splits* of our data set into these *leaves*.
* The algorithm knows when to stop and when there is a certain minimum information that needs to be added.
* And at certain stage the algorithm cannot add any more information to our *set-up* by *splitting* these *leaves*, then it stops. A value is set which determines when to stop (for example less than 5% of your total points in that leaf and then that leaf wouldn't be created).

So there are different variations and different options. The most important thing is where the splits are happening.

* The final leaves are called *terminal* *leaves*.
* The decision tree: Now we're going to create these *splits* one by one and alongside we're going to actually start drawing our *decision* *tree*.

|  |  |
| --- | --- |
| Splitting Data-set | Decision Tree |
|  |  |
|  |  |
|  |  |
|  | Now that’s our decision tree. |

* How to predict: Next we determine y, the value is determined by taking the average of the values in a terminal-leafs (the last iteration) . i.e. just take the average of ***y*** for all of points of the leaf and that'll be the value that will be assigned to any new point that falls in that terminal.
* The whole point of this exercise is to add more information into our chart into our system to better predict ***y***.
* Our machine learning algorithm just take the average across all of the points and whatever that is wherever you point the new element of data, that is added to our data set wherever it falls.
* So, we've split our diagram up into terminal leaves and the machine learning algorithm has added information nurture into our system. So that we can more accurately predict the value or assign the value of ***y*** to a new coming element. Following is our *completed* *decision* *tree*.



**2.5.2 Decision Tree Regression Model practice with Python**

Practiced version

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Position\_Salaries.csv")

X = dataSet**.**iloc[:, 1:2]**.**values

y = dataSet**.**iloc[:, 2]**.**values

#*# Feature-Scaling*

#*from sklearn.preprocessing import StandardScaler*

#*sc\_x = StandardScaler()*

#*sc\_y = StandardScaler()*

#*X\_scaled = sc\_x.fit\_transform(X)*

#*y\_scaled = sc\_y.fit\_transform(y.reshape(-1, 1))*

#*Data Split : No need for this example*

#*Fit dataset to model*

**from** sklearn**.**tree **import** DecisionTreeRegressor

regressor = **DecisionTreeRegressor**(random\_state= 0)

regressor**.fit**(X, y)

#*Predict*

y\_pred = regressor**.predict**([[6.5]])

**print**("The predicte value ffor 6.5 is : ", y\_pred)

#*plot the model*

pLt**.scatter**(X, y, color = "red")

pLt**.plot**(X, regressor**.predict**(X), color = "green")

pLt**.title**("Truth or Bluff (Decision Tree Regression)")

pLt**.xlabel**("Position level")

pLt**.ylabel**("Salary")

* Prediction: Our prediction is calculated by

y\_pred = regressor**.predict**([[6.5]])

* Which gives 150,000. This prediction is below 160,000 (the new employee asks). This is not a good prediction, actually its calculated from the mean-value.

The new trap: We get the following figure: In the algorithm of decision tree regression, the entropy in the information increases by ***splitting the independent variables into several intervals***. In the intuition we had two independent variables and . In this example we have *only one independent variable*.



So according to algorithm it's taking the average in each interval. So either there are *infinite* *intervals* so that the above graph is so *smooth* or we have problem with our ***matplot*** settings. Of course *Decision Tree algorithm* not considering infinite intervals, so it’s the issue with our plotting.

In our previous template, it's only plotting the predictions of the 10 salary's corresponding to the 10 levels and then its joining the predictions by a straight line here because it had no predictions to plot in this interval.

* Nonlinear and non-continuous regression model: Now we're facing a new kind of regression model. It's the nonlinear and non-continuous regression model.
* Indeed all the previous regressions were Linear and they were all continuous. But here the decision tree regression model is not continuous and this is the first non-continuous machine model for us.

|  |  |
| --- | --- |
| * Visualize the non-continuous regression model: When we visualize the results for Higher Resolution and smoother curve we can observe the mean-values and non-continuous regression model. Following is the *Decision Tree algorithm* in One-dimension.   X\_grid = np**.arange**(min(X), max(X), 0.01)  X\_grid = X\_grid**.reshape**(len(X\_grid), 1) #*reshape matrix/array*  pLt**.scatter**(X, y, color = "red")  pLt**.plot**(X\_grid, regressor**.predict**(X\_grid), color = "green")  pLt**.title**("Truth or Bluff (Decision Tree Regression)")  pLt**.xlabel**("Position level")  pLt**.ylabel**("Salary")  pLt**.show**() |  |

Use The high resolution

#*plot the model*

#*pLt.scatter(X, y, color = "red")*

#*pLt.plot(X, regressor.predict(X), color = "green")*

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

X\_grid = X\_grid**.reshape**(len(X\_grid), 1) #*reshape matrix/array*

pLt**.scatter**(X, y, color = "red")

pLt**.plot**(X\_grid, regressor**.predict**(X\_grid), color = "green")

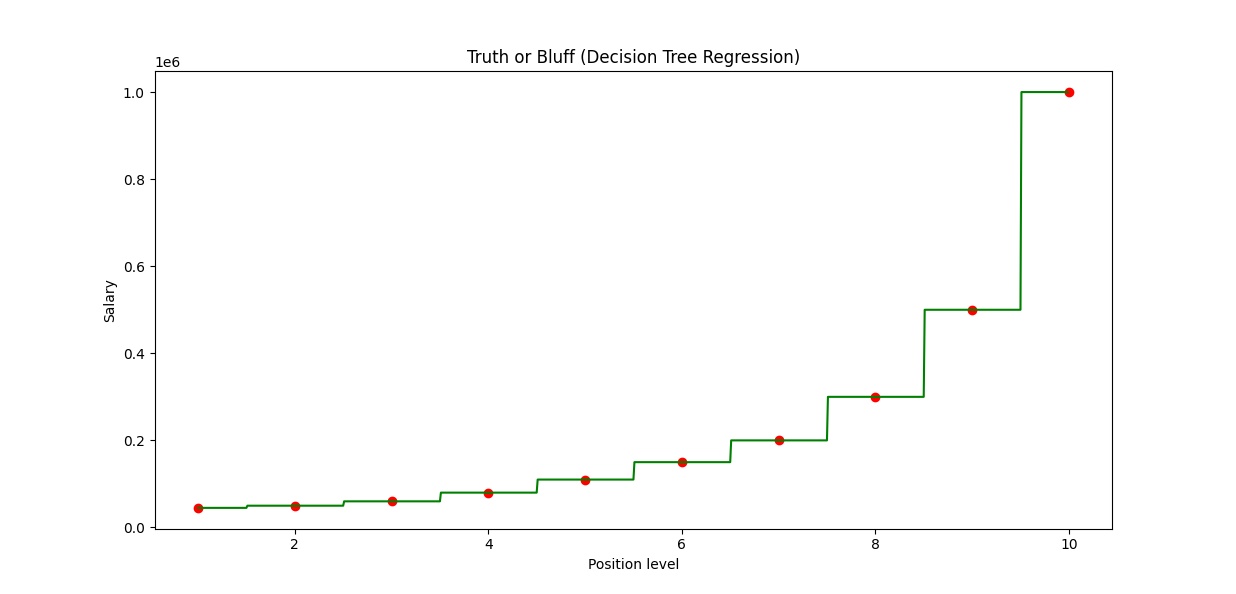
pLt**.title**("Truth or Bluff (Decision Tree Regression)")

pLt**.xlabel**("Position level")

pLt**.ylabel**("Salary")

pLt**.show**()

* We can see the vertical line below as we set resolution set to ***0.01*** so the vertical lines makes it Non-Continuous.



* Conclusion: the decision tree regression model is not an *interesting* model in *one-dimension* but it can be a very interesting and very powerful model in more dimensions.
* About Random Forrest: Random Forrest is actually a team of several decision trees. Previous example is the result of one tree what do you think we'll get with a team of 10 trees or even a hundred trees or 500 trees?

**2.5.3 FAQ**

* How does the algorithm Split the Data points?
* It uses reduction of standard deviation of the predictions. In other words, the *standard deviation* is *decreased* right after a split. Hence, building a decision tree is all about finding the attribute that returns the ***highest standard deviation reduction*** (i.e., the most homogeneous branches).
* What is the Information Gain and how does it work in Decision Trees?
* The Information Gain in Decision Tree Regression is exactly the Standard Deviation Reduction we are looking to reach. We calculate by how much the ***Standard*** ***Deviation*** ***decreases*** after each split. Because the more the *Standard Deviation* is *decreased* after a *split*, the more homogeneous the child nodes will be.
* What is the Entropy and how does it work in Decision Trees?
* The Entropy measures the disorder in a set, here in a part resulting from a split. So the ***more homogeneous*** is your data in a part, the ***lower*** will be the ***entropy***. The more you have splits, the more you have chance to find *parts* in which your *data* is *homogeneous*, and therefore the *lower* will be the *entropy* (close to 0) in these parts. However you might still find some nodes where the *data is not homogeneous*, and therefore the *entropy* would *not be that small*.
* Does a Decision Tree make much sense in 1D?
* Not really, as we saw in the practical part of this section. In 1D (meaning *one* *independent* *variable*), the *Decision* *Tree* clearly tends to *overfit* the data. The *Decision* *Tree* would be much more relevant in *higher* *dimension*, but keep in mind that the implementation we made here in 1D would be exactly the same in *higher* *dimension*. Therefore you might want to keep that model in your *toolkit* in case you are dealing with a *higher* *dimensional* *space*. This will actually be the case in Chapter 3 - Classification, where we will use Decision Tree for Classification in 2D, which you will see turns out to be more relevant.

[Decision Tree Regression in R

Why do we get different results between Python and R?

The difference is likely due to the random split of data. If we did a cross-validation (see Part 10) on all the models in both languages, then you would likely get a similar mean accuracy. That being said, we would recommend more using Python for Decision Trees since the model is slightly better implemented in Python.]

* Is the Decision Tree appropriate here?
* Here in this example, we can clearly see that the fitting curve is a stair with large gaps in the discontinuities. That decision tree regression model is therefore not the most appropriate, and that is because we have only one independent variables taking discrete values. So what happened is that the prediction was made in the lower part of the gap in Python, and made in the upper part of the gap in R. And since the gap is large, that makes a big difference. If we had much more observations, taking values with more continuity (like with a 0.1 step), the gaps would be smaller and therefore the predictions in Python and R far from each other.
* No p-value for Decision Tree ?
* You couldn’t use p-value because ***Decision*** ***Tree*** is not a ***linear*** ***model***, and p-values apply only to linear models. Therefore feature selection is out of the question. But you could do feature extraction, which you will see in *Chapter 9 - Dimensionality Reduction*. That you can apply to *Decision Trees*, and it will *reduce the number of your features*.