DATA SCIENCE

Decision Tree Regressor explained in depth

GEORGIOS DRAKOS
23 MAY 2019 • 17 MIN READ

You've successfully subscribed to GDCoder!

environment. Decision Tree can be used both in classification and regression problem.This article present the Decision Tree Regression Algorithm along with some advanced topics.

Table of Contents

Photo by Todd Quackenbush / Unsplash

Note: If you are interested to learn how Random Forest Regressor works check my article here.

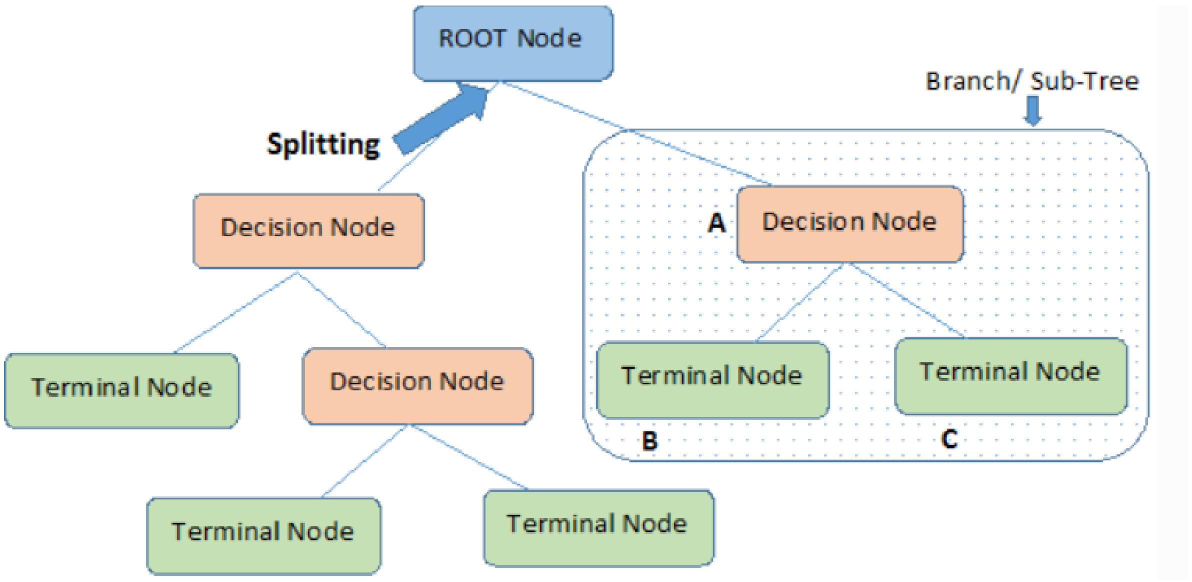
Introduction

Decision Trees can be summarized with the below bullet points:

- Decision trees are predictive models that use a set of binary rules to calculate a target value.
- Each individual tree is a fairly simple model that has branches, nodes and leaves.

Before diving into let's look at the basic terminology used with decision further gets divided into two or more homogeneous sets.

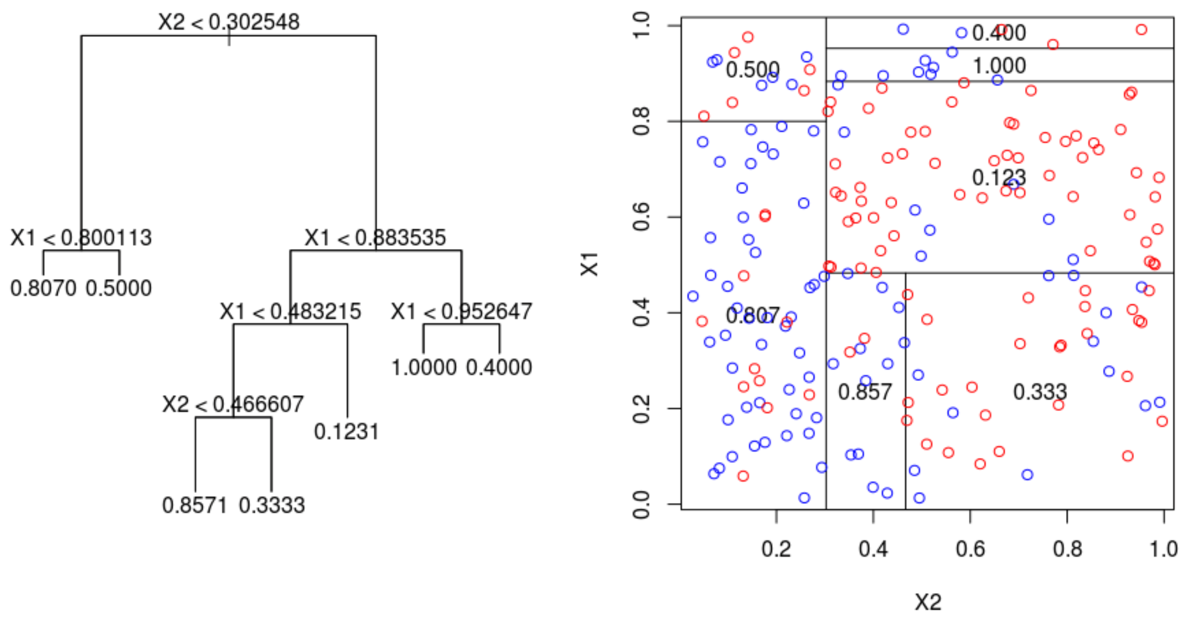
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.



?How does it work?

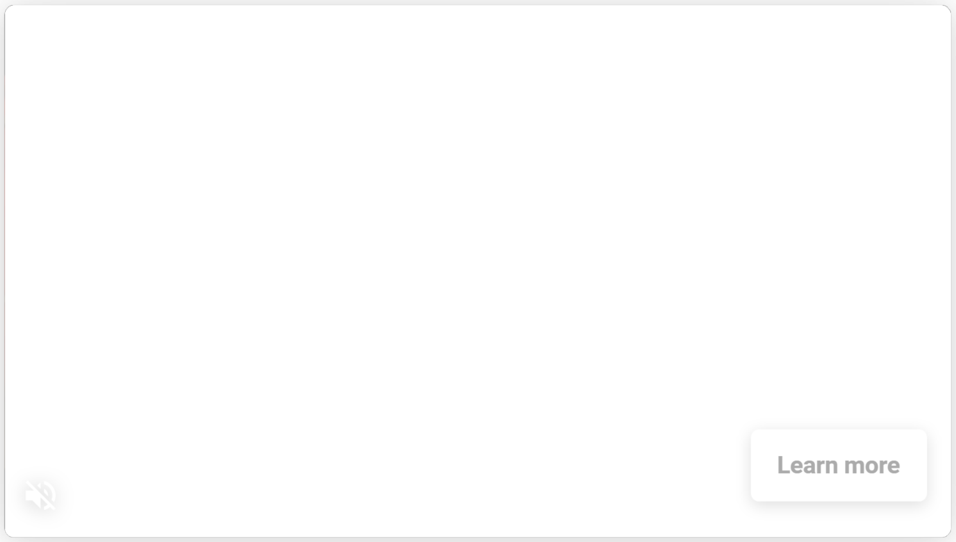
A decision tree is arriving at an estimate by asking a series of questions to the data, each question narrowing our possible values until the model get confident enough to make a single prediction. The order of the question as well as their content are being determined by the model. In addition, the questions asked are all in a True/False form.

This is a little tough to grasp because it is not how humans naturally think, and perhaps the best way to show this difference is to create a real decision tree from. In the above problem x1, x2 are two features which allow us to make predictions for the target variable y by asking True/False questions.



For each True and False answer there are separate branches. No matter the answers to the questions, we eventually reach a prediction (leaf node).

One aspect of the decision tree I should mention is how it actually learns (how the ‘questions’ are formed and how the thresholds are set). As a supervised machine learning model, a decision tree learns to map data to outputs in what is called the training phase of model building.



During training, the model is fitted with any historical data that is relevant to the problem domain and the true value we want the model to learn to predict. The model learns any relationships between the data and the target variable.

After the training phase, the decision tree produces a tree similar to the one shown above, calculating the best questions as well as their order to ask in order to make the most accurate estimates possible. When we want to make a prediction the same data format should be provided to the model in order to make a prediction. **The prediction will be an**

How is Splitting Decided for Decision Trees?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.Decision trees regression normally use mean squared error (MSE) to decide to split a node in two or more sub-nodes.
Suppose we are doing a binary tree the algorithm first will pick a value, and split the data into two subset. For each subset, it will calculate the MSE separately. The tree chooses the value with results in smallest MSE value.

Let's examine how is Splitting Decided for Decision Trees Regressor in more details. The first step to create a tree is to create the first binary decision. How are you going to do it?

- We need to pick a variable and the value to split on such that the two groups are as different from each other as possible.
- For each variable, for each possible value of the possible value of that variable see whether it is better.
- How to determine if it is better? Take weighted average of two new nodes (mse*num_samples)

To sum up, we now have:

- A single number that represents how good a split is which is the weighted average of the mean squared errors of the two groups that create

try every possible value of that variable and see which variable and which value gives us a split with the best score.

some stopping condition (defined by hyperparamters) is met:

- When you hit a limit that was requested (for example `max_depth`)
- When your leaf nodes only have one thing in them (no further split is possible, MSE for the train will be zero but will overfit for any other set -not a useful model)

Questions and Answers

Are there circumstances when it is better to split into 3 groups ?

It is never necessary to do more than one split at a level because you can just split them again.

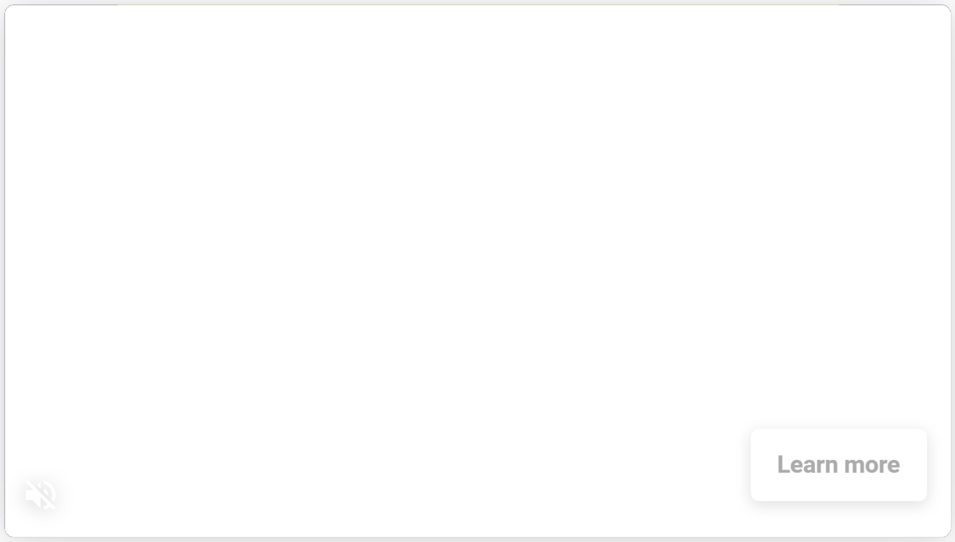
How it makes predictions?

Given a data point you run it through the entirely tree asking True/False questions up until it reaches a leaf node. The final prediction is the average of the value of the dependent variable in that leaf node.

Since we try every variable and every possible value of that variable to decide a split. Would the training

second and it has multiple cores and each core has something called **single instruction, multiple data** (**SIMD**) where it can do up to eight computations per core at once. In addition if a GPU is being used the process is even faster since its performance is measured in teraflops so trillions of floating-point operations per second. So this is where when it comes to designing algorithms it's very difficult for us mere humans to realize how stupid algorithms should be given how fast today's computers are. To sum up there are quite a few operations but that trillions per second you hardly notice it.

The underlying model is simply the average of the data points. For the instead of using the average to use median or we can even run a linear regression model. There are a lot of things we could do but in practice the average works really well.They do exist random forests models where the leaf nodes are independent linear regressions but they're not widely used.



A DT with depth n it will contain the DT with depth n-1 given the same data, hyperparameters?

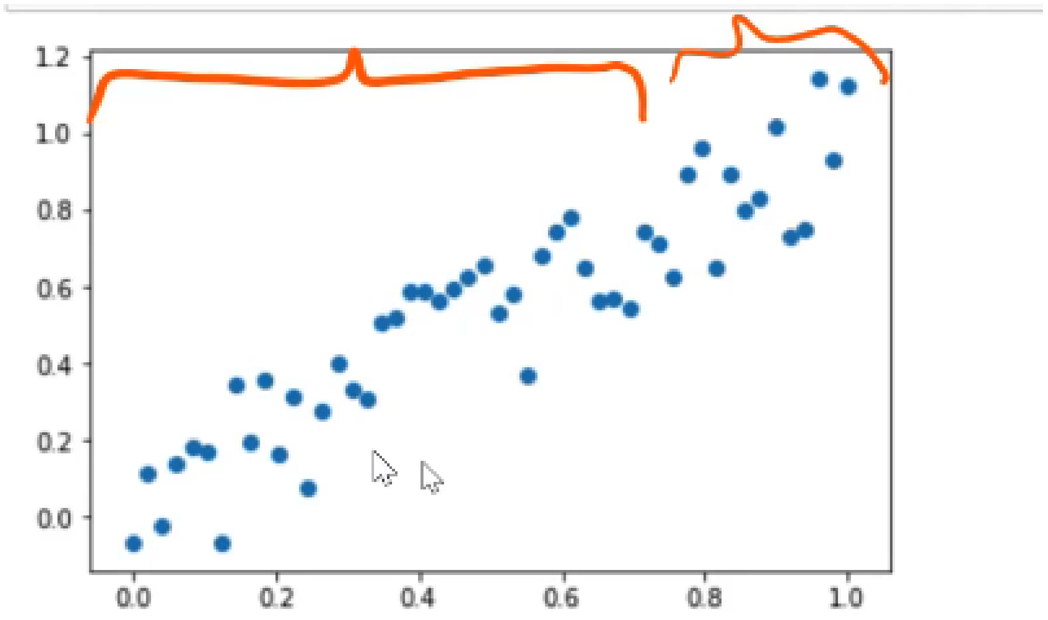
So if we create a tree with a depth of three and another one where we get rid of the max depth, the tree without the max depth constraint will contain the tree with the depth of three. This assumes that the data, hyperparameters will be the same and no randomness have been added (the deeper tree will contain the less deep tree).

❌ When not to use?

As we saw previously, a decision tree will be unable to make accurate

argument with an example:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
x = np.linspace(0,1)
y = x + np.random.uniform(-0.2,0.2,x.shape)
plt.scatter(x,y)
```



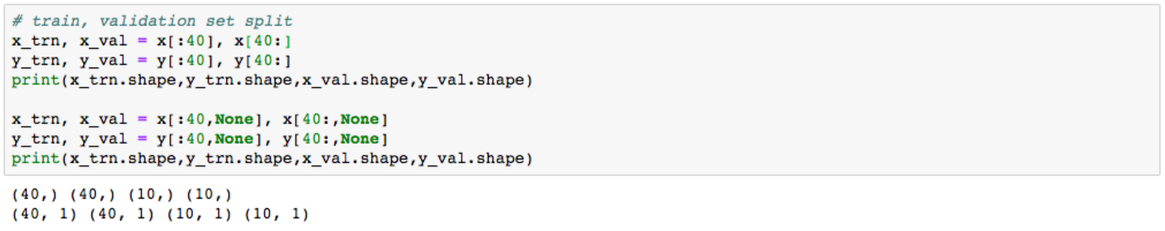
We're now going to build a decision tree model and what kind of acts as if this is a time series problem. So I'm going to take left part as a training set. And take the right part as our validation set.

```
# train, validation set split
x_trn, x_val = x[:40,None], x[40:,None]
```

```
m = DecisionTreeRegressor(max_depth=6).fit(x_trn, y_trn)
```

Learn more

Please note that DecisionTreeRegressor expect a 2D array (or an array of rank 2) and not an 1D array into a 2D array.



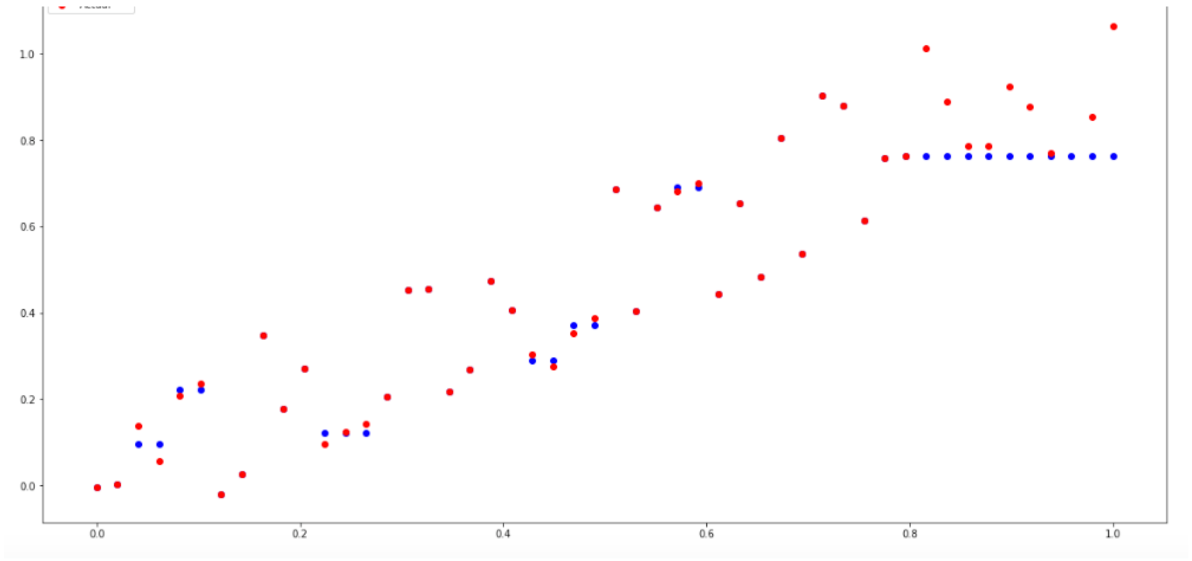
```
# train, validation set split
x_trn, x_val = x[:40], x[40:]
y_trn, y_val = y[:40], y[40:]
print(x_trn.shape,y_trn.shape,x_val.shape,y_val.shape)

x_trn, x_val = x[:40,None], x[40:,None]
y_trn, y_val = y[:40,None], y[40:,None]
print(x_trn.shape,y_trn.shape,x_val.shape,y_val.shape)
```

```
(40,) (40,) (10,) (10,)
(40, 1) (40, 1) (10, 1) (10, 1)
```

Let's now plot the result:

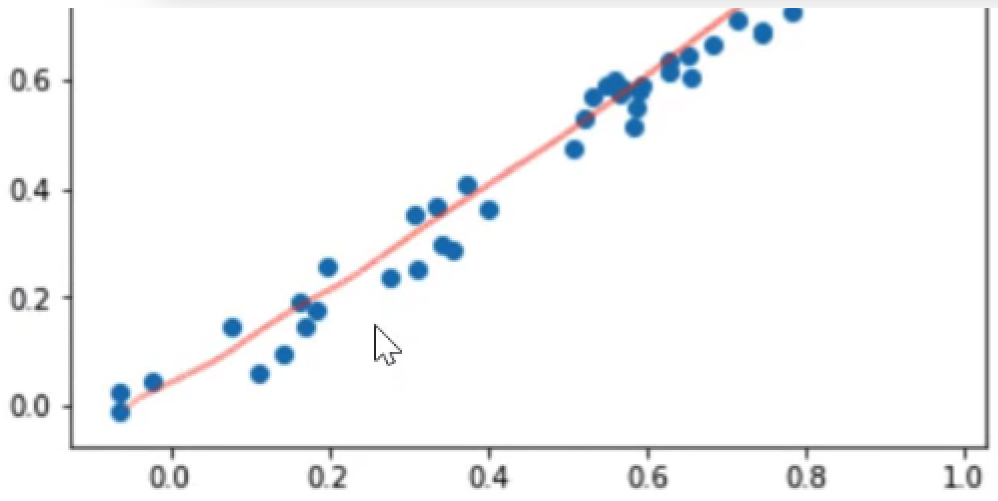
```
plt.scatter(x_val,m.predict(x_val),color='blue',label='Prediction')
```

```
plt.legend(loc='upper left')
```



If you don't know how decision trees work then this is going to render the model useless. In other words, decision tree and tree based models in general are unable to extrapolate to any kind of data they haven't seen before, particularly future time period as it's just averaging data points it has already seen.

In a nutshell Decision trees and tree based models in general just do a clever nearest neighbours.



One major disadvantage of Decision Trees is that they are prone to overfitting. That's why they are rarely used and instead other tree based models are preferred like Random Forest and XGBoost.

To sum up, Decision tree and general tree based methods are a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems. As noted above they are used in regression problems if and only if the target variable is inside the range of

Advantage & Disadvantages of Decision Trees

Photo by Johann Siemens / Unsplash

Advantages

1. **It can be used for both Classification and Regression problems**
2. **Easy to Understand, Interpret, Visualise**
3. **Useful in Data exploration:** Decision tree is one of the fastest

of more variables. With the help of decision trees, we can create new variables / features that has better power to predict target exploration stage.

4. **Less data preparation required:** It is not influenced by outliers and missing values to a fair degree.
5. **Data type is not a constraint:** It can handle both numerical and categorical variables.
6. **Non Parametric Method:** Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.
7. **Can capture Nonlinear relationships**

Disadvantages

1. **Over fitting:** Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
2. **Not fit for continuous variables**: While working with continuous numerical variables, decision tree looses information when it categorizes variables in different categories.
3. **Cannot extrapolate.**
4. **Decision trees can be unstable:** Small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and **boosting**.
5. **No Guarantee to return the globally optimal decision tree**.

data science today, I highly recommend the below video course from Udacity:

Learn to Become a Data Scientist Online | Udacity | Udacity

Gain real-world data science experience with projects from industry experts. Take the first step to becomin...

Udacityicon-checkmarkicon-check...

UDACITY

While for book lovers:

- **"Python for Data Analysis"** by Wes McKinney**,** best known for creating the Pandas project.
- **"Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow"** by Aurelien Geron, currently ranking first in the best sellers Books in AI & Machine Learning on Amazon.
- **"Deep Learning"** by **Ian Goodfellow** research scientist at OpenAI.

Techniques to avoid over-fitting

Often you may find that you've overfitted your model to the data, which is often detrimental to the model's performance when you introduce new

You've successfully subscribed to GDCoder!

each observation. Thus, preventing overfitting is of major importance when training a decision tree and it can be done in 2 ways:

2. Tree pruning
3. Use a Random Forest

Let's discuss both of these briefly.

Setting Constraints on Tree Size

This can be done by fine-tuning some hyperparameters which are used to define a tree. First, let's look at the general structure of a decision tree:

tree infographic

By understanding the role of parameters used in tree modeling will help you to better fine-tuned a decision tree both in R & Python.

Minimum samples for a node split

- Defines the minimum number of samples (or observations) which are required in a node to be considered for splitting.
- Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.
- Too **high values** can lead to **under-fitting** hence, it should be fine-tuned with care.

Minimum samples for a terminal node (leaf)

terminal node or leaf.

- Too **high values** can lead to **under-fitting** hence, it should be fine tuned with care.

Maximum depth of the tree (vertical depth)

- Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.
- Too **high values** can lead to **over-fitting** hence, it should be fine-tuned with care.

Maximum features to consider for a split

- The number of features to consider while searching for the best split. These will be randomly selected.
- As a thumb-rule, square root of the total number of features works great but we should check up to 30-40% of the total number of features.
- Higher values can lead to over-fitting but depend on case to case.

Tree Pruning

As discussed earlier, the technique of setting constraint is a greedy-approach. In other words, it will check for the best split instantaneously and move forward until one of the specified stopping condition is reached.

data and poorly generalizing to new samples. A small tree might not capture important structural information about the sample space.

decrease error. This problem is known as the horizon effect. A common strategy is to grow the tree until each node contains a small number of instances then use pruning to remove nodes that do not provide additional information.

Pruning should reduce the size of a learning tree without reducing predictive accuracy as measured by a cross-validation set. There are many techniques for tree pruning that differ in the measurement that is used to optimize performance.

One of the simplest forms of pruning is reduced error pruning. Starting at the leaves, each node is replaced with its mean value. If the MSE is not affected then the change is kept. While somewhat naive, reduced error pruning has the advantage of **simplicity and speed**.

Use Random Forest

🥊 Decision Trees Vs Random Forests

Photo by D. Jameson RAGE / Unsplash

The common argument for using a decision tree over a random forest is that decision trees are easier to interpret, you simply look at the decision tree logic. However, in a random forest, you're not going to want to study the decision tree logic of 500 different trees.

At the same time, Random Forest is actually a collection of Decision Trees once they are trained. this is due to the fact that it has to run predictions on each individual tree and then average their predictions to create the final prediction. (a solution to that can be to use cluster fitting different trees at the same time)

A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.

Finally, Decision Trees do suffer from overfitting while Random Forest can prevent overfitting resulting in better prediction most of the time. This is a significant advantage of the Random Forest regression model fact which makes it more appealing to a lot of data scientists.

Hyperparameters of Sklearn Decision Tree

min_samples_leaf : int, float, optional (default=1)

It is the minimum number of samples for a terminal node that we discuss above.

- If int, then consider min_samples_leaf as the minimum number.
- If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.

min_samples_split : int, float, optional (default=2)

It is the minimum samples for a node split that we discuss above.

- If int, then consider min_samples_split as the minimum number.
- If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of

The number of features to consider when looking for the best split:

- If int, then consider max_features features at each split. -If float, then max_features is a percentage and int(max_features * n_features) features are considered at each split.
- If "auto", then max_features=sqrt(n_features).
- If "sqrt", then max_features=sqrt(n_features) (same as "auto").
- If "log2", then max_features=log2(n_features).
- If None, then max_features=n_features.

max_depth : integer or None, optional (default=None)

- The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

🆚 Tree-based models Vs Linear models

Actually, you can use any algorithm. It is dependent on the type of problem you are solving. Let's look at some key factors which will help you to decide which algorithm to use:

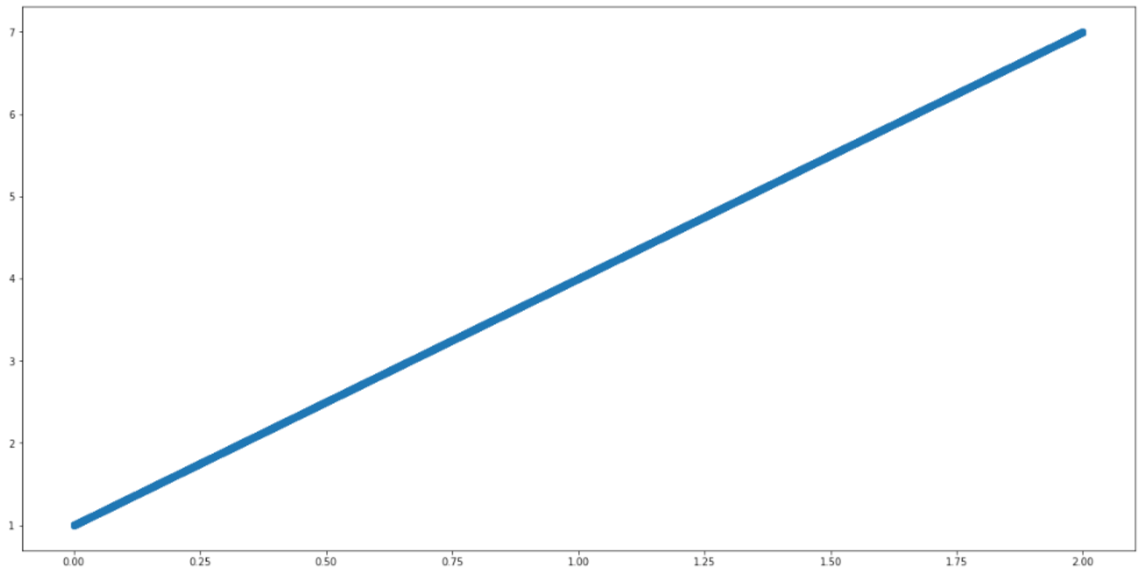
1. If the relationship between dependent & independent variable is well approximated by a linear model, linear regression will outperform tree based model.

classical regression method.

Decision tree models are even simpler to interpret than linear regression!

Let's prove our argument on points 1 and 2 with an example:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
plt.figure(figsize=(20,10))
x = np.linspace(0,2,10000)
y = 1+ 3*x
plt.scatter(x,y)
```

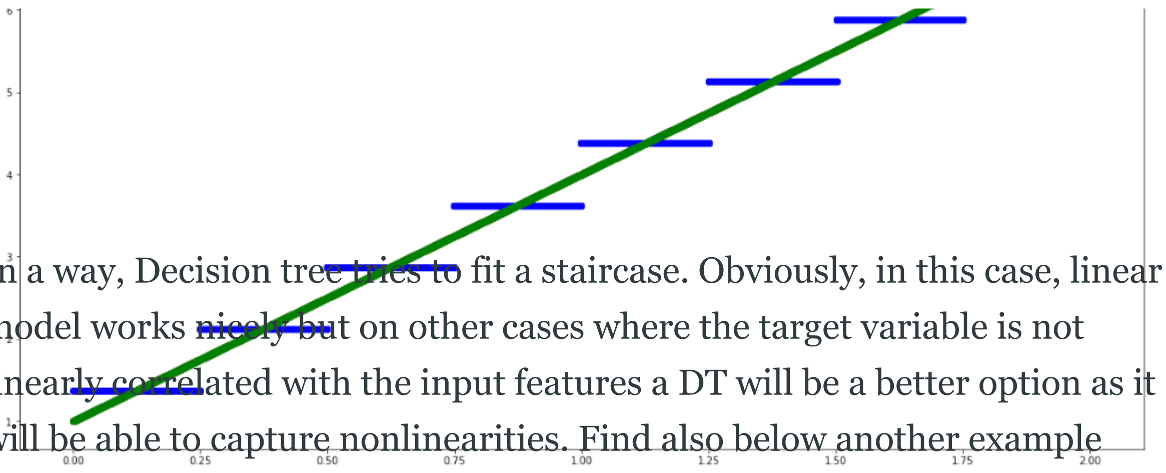


linear model to fit perfectly as the target variable is linearly related to

```
idxs_train = sorted(np.random.permutation(len(x))[:int(0.7*len(x))])
idx_test = [i for i in range(0,len(x)) if i not in idxs_train]

# train, validation set split
x_trn, x_val = x[idxs_train,None], x[idx_test,None]
y_trn, y_val = y[idxs_train,None], y[idx_test,None]


# fit a model
dt = DecisionTreeRegressor(max_depth=3).fit(x_trn, y_trn)
l = LinearRegression().fit(x_trn, y_trn)
```

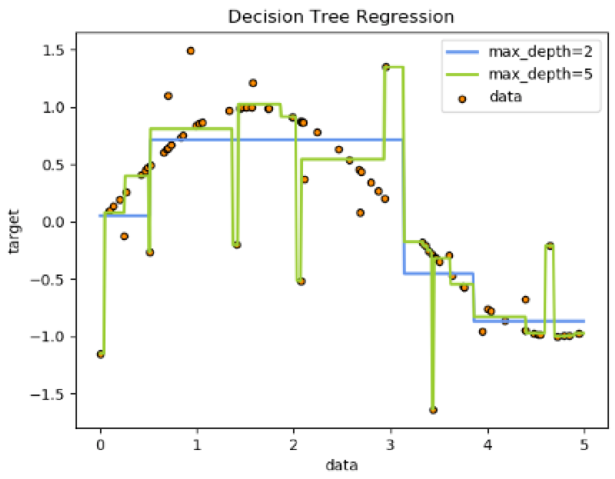
Please note that both models expect a 2D array (or an array of rank 2) and not a 1D array into a 2D array. Let's now plot the result:

```
plt.figure(figsize=(20,10))
plt.scatter(x,dt.predict(x[:,None]),color='blue',label='Prediction DT')
plt.scatter(x,l.predict(x[:,None]),color='green',label='Prediction Linear'
plt.scatter(x,y,color='red',label='Actual')
plt.legend(loc='upper left')
```



In a way, Decision tree tries to fit a staircase. Obviously, in this case, linear model works nicely but on other cases where the target variable is not linearly correlated with the input features a DT will be a better option as it will be able to capture nonlinearities. Find also below another example from scikit-learn documentation:

1.10.2. Regression



Decision trees can also be applied to regression problems, using the `DecisionTreeRegressor` class.

As in the classification setting, the fit method will take as argument arrays X and y, only that in this case y is expected to have floating point values instead of integer values:

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```

You've successfully subscribed to GDCoder!

📊 Visualise a Decision Tree model

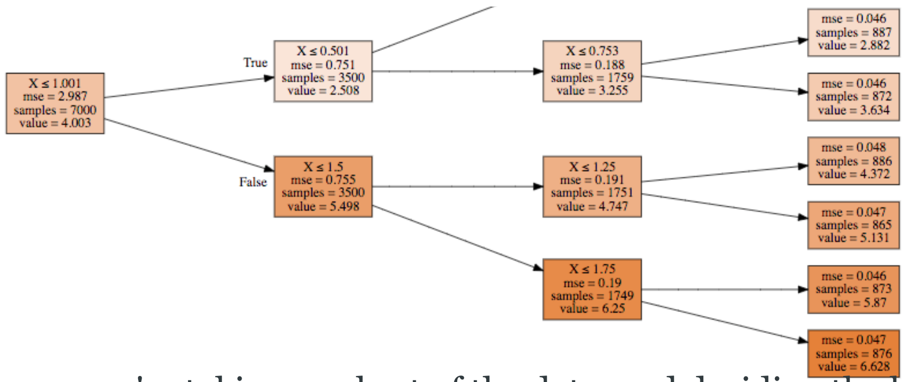
Visualising a Decision Tree is fairly simple. Let's draw the decision tree that was trained above.

First, we will need to import some additional libraries:

```
from sklearn.tree import export_graphviz
import IPython, graphviz, re, math
```

And then we can simply draw the Decision Tree as below:

```
def draw_tree(t, col_names, size=9, ratio=0.5, precision=3):
    """ Draws a representation of a random forest in IPython.
    Parameters:
    -----------
    t: The tree you wish to draw
    df: The data used to train the tree. This is used to get the names of
    """
    s=export_graphviz(t, out_file=None, feature_names=col_names, filled=Tr
                      special_characters=True, rotate=True, precision=prec
    IPython.display.display(graphviz.Source(re.sub('Tree {',
       f'Tree {{ size={size}; ratio={ratio}',s)))
col_names =['X']
draw_tree(dt, col_names, precision=3)
```



As you can see we're taking a subset of the data, and deciding the best manner to split the subset further. Our initial subset was the entire data set, and we split it according to the rule `X<=1.001`. Then, for each subset, we performed additional splitting until we were able to correctly predict the target variable while respecting the constraint of `max_depth=3`.

Conclusion

Decision trees are one of the most widely-used machine learning models, due to the fact that they work well with noisy or missing data and can easily be ensembled to form more robust predictors. Moreover, you can directly visual your model's learned logic, which means that it's an incredibly popular model for domains where model interpretability is important.

. . .

Thanks for reading, if you liked this article, please consider subscribing to my blog. That way I get to know that my work is valuable to you and also

As always keep studying, keep creating

◉ References

- https://scikit-learn.org/stable/modules/tree.html
- https://en.wikipedia.org/wiki/Decision_tree_pruning
- https://stackoverflow.com/questions/49428469/pruning-decision-trees

GDCoder – Decision Tree Regressor explained in depth

DATA SCIENCE

What is a Recurrent Neural Networks (RNNS) and Gated Recurrent Unit (GRUS)

Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many sequential data and among others used by Apples Siri and Googles Voice Search. Their great advantage

GEORGIOS DRAKOS
23 MAY 2019 • 7 MIN READ

INTERVIEW CODING PROBLEMS

Interview Coding Problems: 1. Find the first missing positive integer of a list 2. Challenge of using closures to store data 3.Given

You've successfully subscribed to GDCoder!

types of challenges and puzzles can help you become a better problem solver, learn the

GDCoder – Decision Tree Regressor explained in depth

Latest Posts Facebook Twitter Ghost