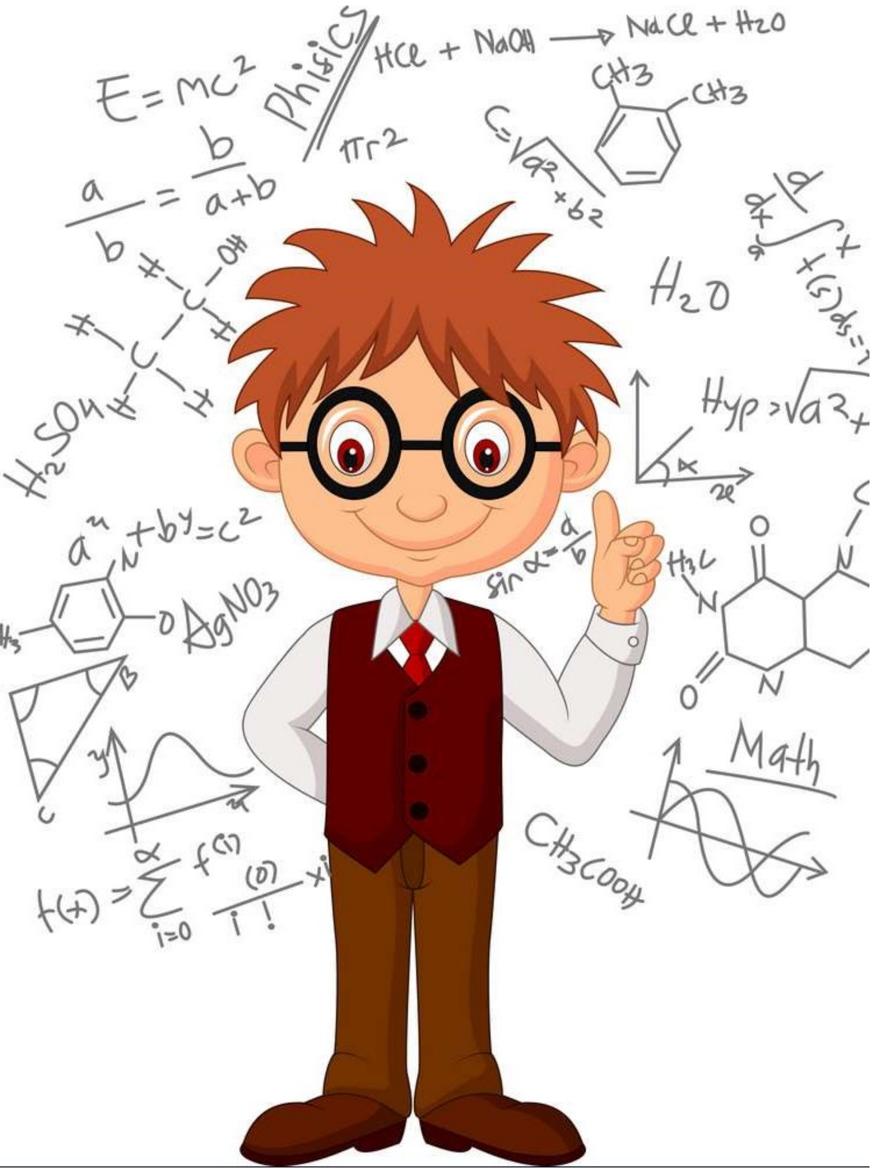
Predictive Data Analytics

Lecturer: Marina Iantorno

E-mail: miantorno@cct.ie

July 2022

cct | College Dublin
Computing • IT • Business



In today's class we will cover:

- ❑ Support Vector Regression
- ❑ Regression Tree
- ❑ Practice in Python

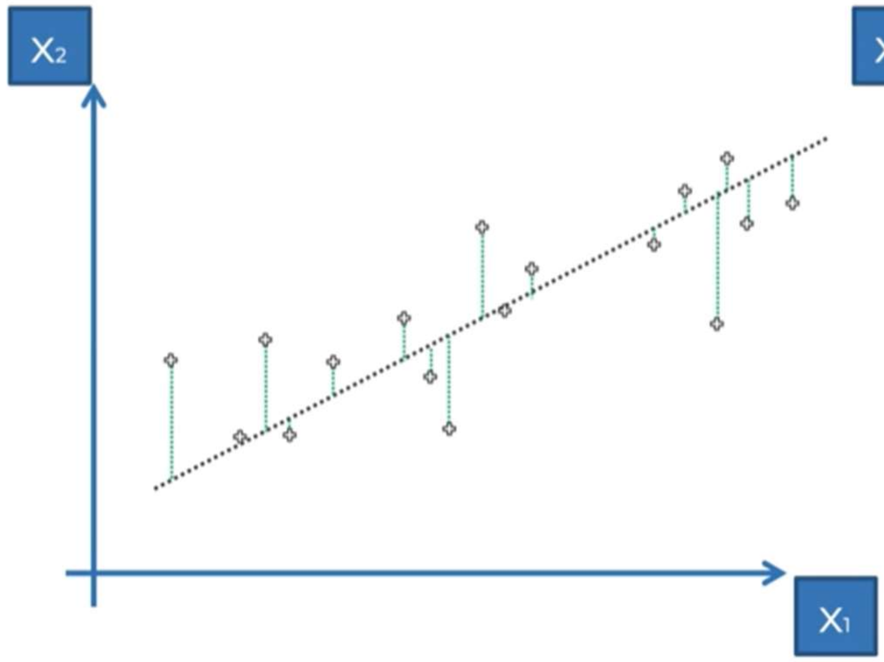
Regression

SUPPORT VECTOR REGRESSION

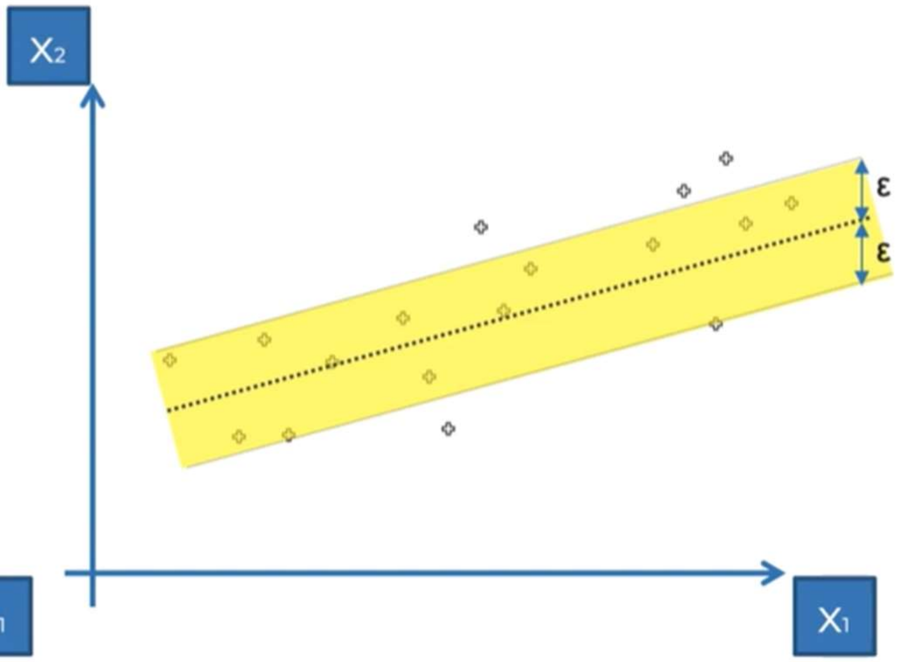
SVR

There is a very common term in ML called “Support Vector Machine”. This is a supervised learning model that analyses data for classification and regression analysis. When we work with regression models, we use Support Vector Regression, a powerful algorithm created by Vladimir Vapnik and his colleagues between 1990. Let’s see an example on how to use it and what is the intuition we could get from it.

SVR

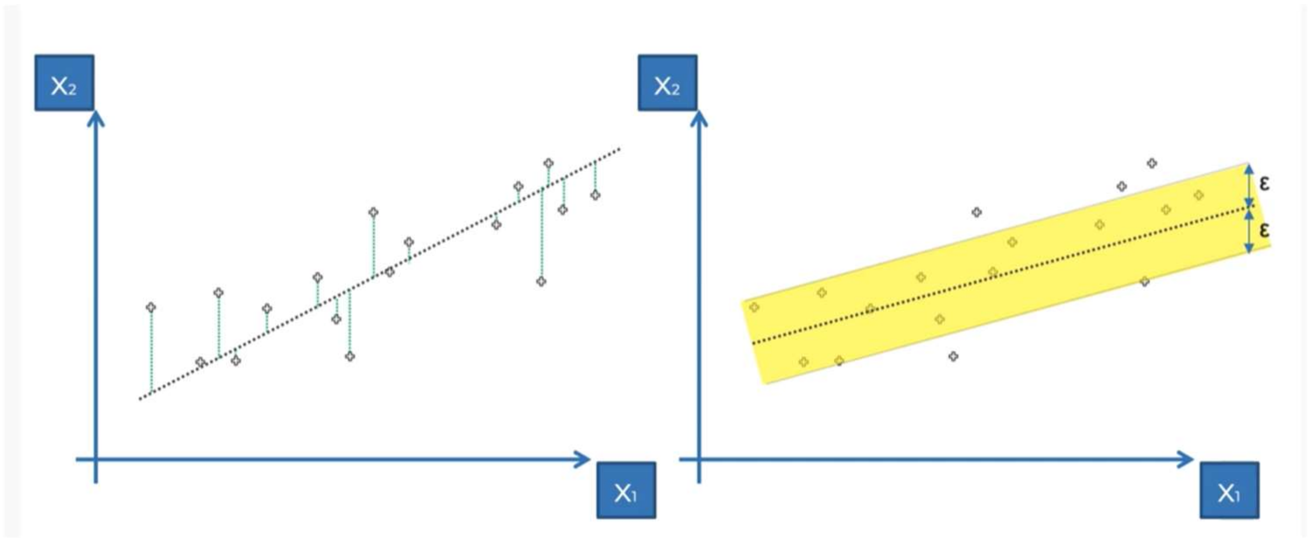


Simple Linear Regression



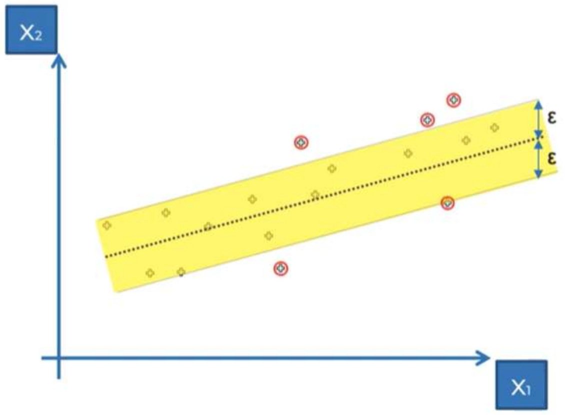
Support Vector Regression

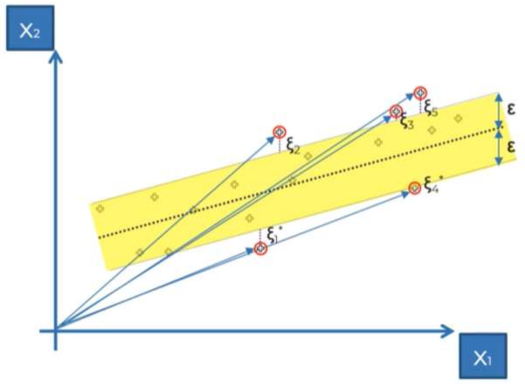
SVR



We saw in previous classes that when we work with simple linear regression, we try to find the line that best fits between the datapoints, and we take into consideration the error that is the distance between the datapoint and the line. Now, the SVR presents a tube that includes the datapoints and the line in the middle. This tube is known as "Insensitive Tube". This is the name because the tube is "insensitive" to the errors. We disregard the errors, and we set a margin of error that we allowed our model to have. We will not stop on the distance between the datapoints and the line.

SVR



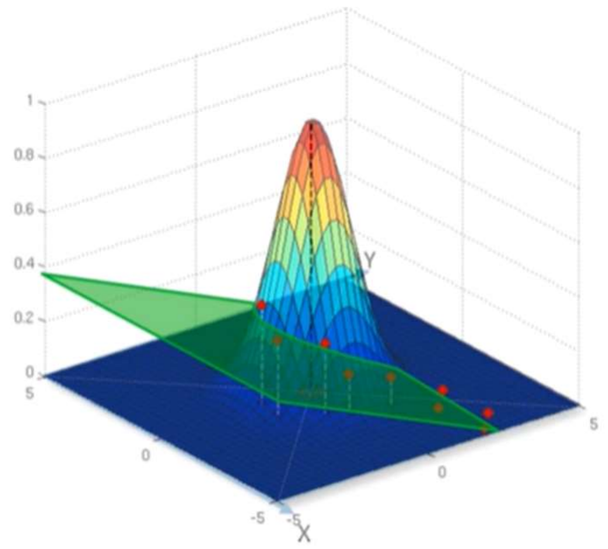


The datapoints that are out of the tube will indicate us the errors, and they are known as “Slack Variables”.

In any case, they are already allowed or expected in our model, and each of the variables will give place to a new vector in our model. These vectors are actually supporting the structure of our tube, and we could call the, supportive vectors. This is the reason to call this model Support Vector Regression.

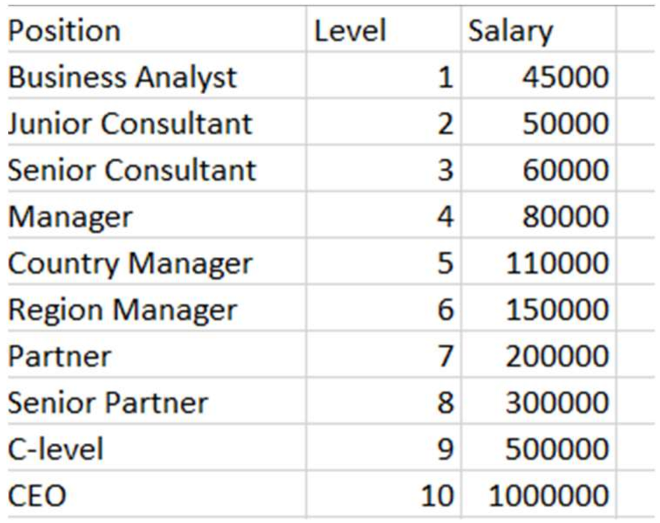
SVR

An important observation is that this model does not work only for Linear Models. In fact, there are Non- Linear SVR, and it is very common to use it there. In this case, we will have a tridimensional graph instead of using only X and Y. We will today an example with a non-linear function, and we will cover in the coming classes multiples examples to understand in general SVM.



SVR

We will try this in Python using the dataset “Position_Salaries.csv”, which is non-linear. We will check if the SVR performs better than the Polynomial Regression.



Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

SVR

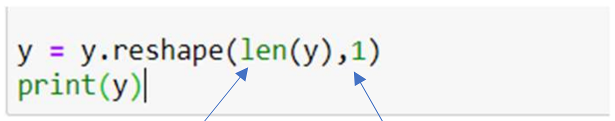
Let’s recap the concept of “Feature Scaling”. When we use it, we standardize the values that we have in our Test set and Train set. We need to do so to ensure that our data is using the same level of measurement. Now, what happens in cases with a small dataset in which we did not split the data into the Training set and Test set? We need to use the Feature Scaling to the whole Y variable, which is the Salary, because we go from 45,000 to 1,000,000 and we need an equilibrium to avoid issues in our results.

SVR

Let’s try it in Python!

SVR

Some outputs from Python.



```
y = y.reshape(len(y),1)
print(y)
```

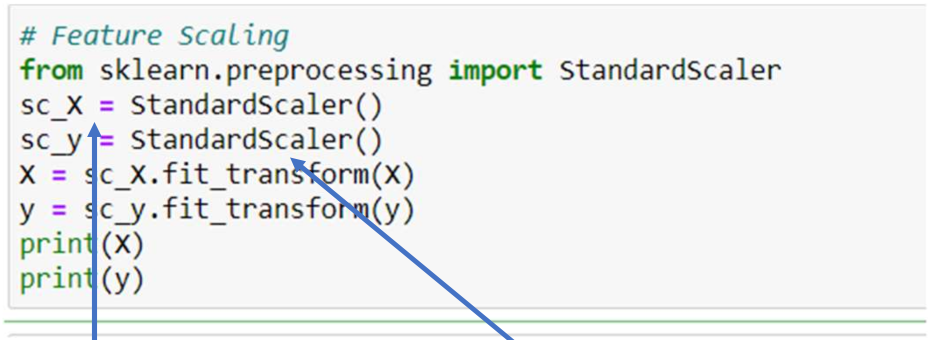
Row length

Number of columns (in this case 1 column0

When we use the reshape function, we will have two columns with the same length of rows. Remember that the first space in the parenthesis belong to the rows and the second one belongs to the columns.

SVR

Some outputs from Python.



```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
print(X)
print(y)
```

We scale the X

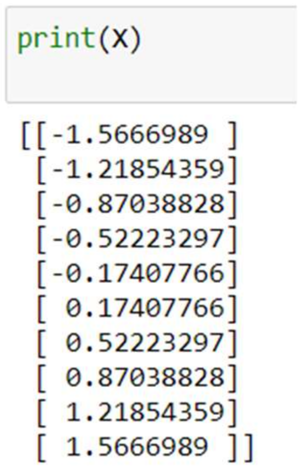
We scale the Y

When we use the Feature Scaling, we try to standardize the values to get them on the same scale, and therefore to have an accurate result.

Remember that when we use a Dummy variable we don't scale them, because they are already designed to work in our model (for example with the codes 0 and 1). In this case, we scale our X variable and the y variable.

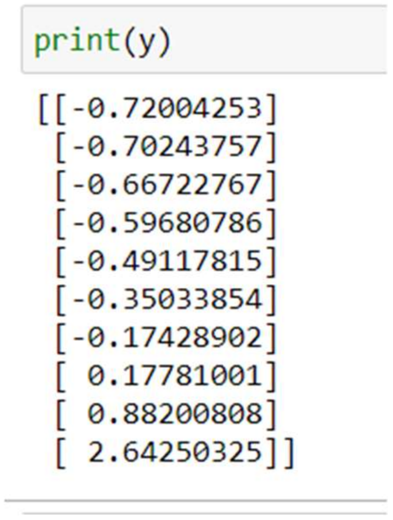
SVR

Some outputs from Python.



```
print(X)

[[-1.5666989 ]
 [-1.21854359]
 [-0.87038828]
 [-0.52223297]
 [-0.17407766]
 [ 0.17407766]
 [ 0.52223297]
 [ 0.87038828]
 [ 1.21854359]
 [ 1.5666989 ]]
```



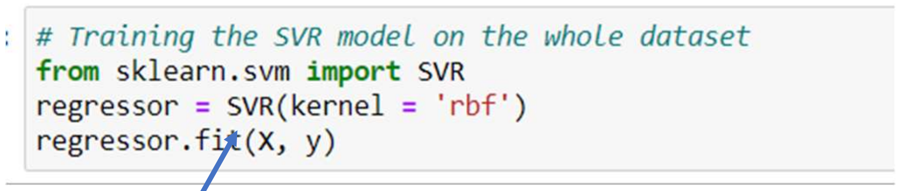
```
print(y)

[[-0.72004253]
 [-0.70243757]
 [-0.66722767]
 [-0.59680786]
 [-0.49117815]
 [-0.35033854]
 [-0.17428902]
 [ 0.17781001]
 [ 0.88200808]
 [ 2.64250325]]
```

This is the print of our variables after the scaling. As we can see, each of the values that we have in the original dataset changed after the scaling. However, those values are closer one another in comparison to the original data in which we saw that 1 was at the same level of 45,000. The standardization is a statistical concept that consists in transform our data to a new array that has a mean of zero and a standard deviation of 1. By doing so, we can have everything on the same scale and having an accurate result in the end.

SVR

Some outputs from Python.



```
# Training the SVR model on the whole dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
```

Our regressor is a SVR

When we use SVR, some common functions we use are the “Kernel Functions”. Kernel is an auxiliar function that will allow us to make calculations in high dimensions on a smooth way.

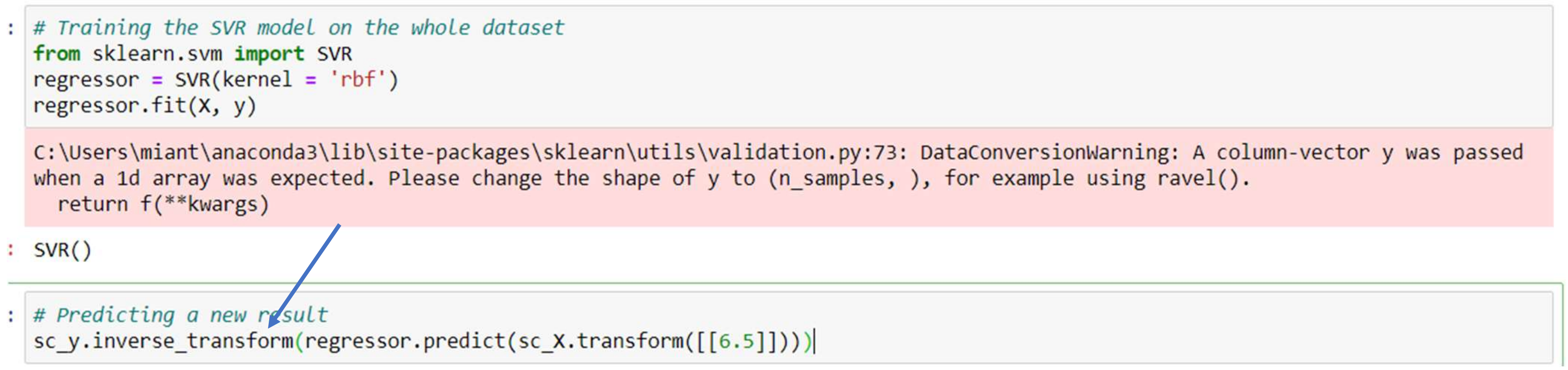
In this case we will use “RBF” which is particularly useful when we have small datasets.

You can read more about the Kernel functions in this link:

https://techvidvan.com/tutorials/svm-kernel-functions/

SVR

Some outputs from Python.



```
# Training the SVR model on the whole dataset
from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
```

```
C:\Users\miant\anaconda3\lib\site-packages\sklearn\utils\validation.py:73: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
```

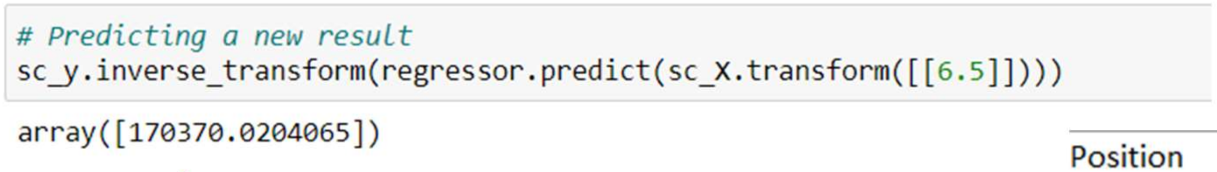
```
SVR()
```

```
# Predicting a new result
sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
```

As we see on the warning, we need to transform our data, and that is why we use the inverse transform, because the variable is located on a different place and a different scale that the models expects. We will use the method called “Inverse Transform Method”.

SVR

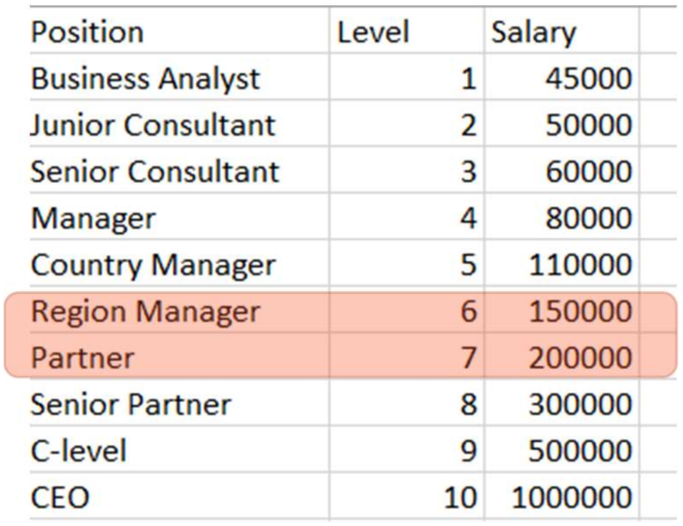
Some outputs from Python.



```
# Predicting a new result
sc_y.inverse_transform(regressor.predict(sc_X.transform([[6.5]])))
```

```
array([170370.0204065])
```

Our prediction is indeed between the level 6 and the level 7.



Position	Level	Salary
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Regression

DECISION TREE

Regression Tree

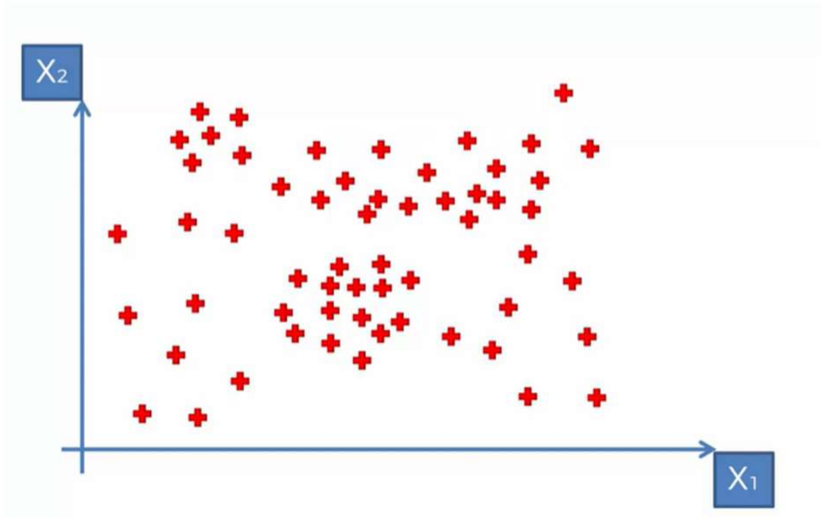
There is a term to define the trees and it is called “CART”. CART stands for Classification And Regression Trees.

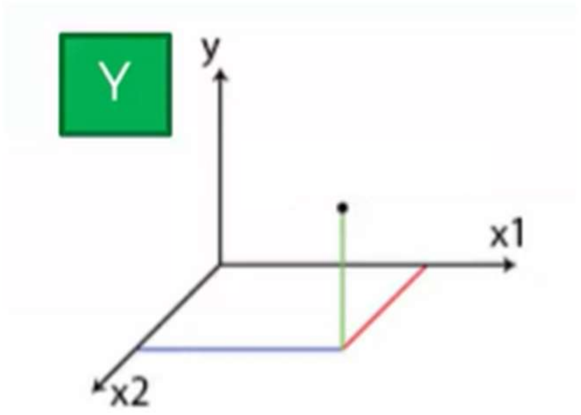
We will focus in today’s class on the Regression Trees.

Imagine that we have two variables, name them X1 and X2, and we have a third variable which is the dependent variable Y.

Regression Tree

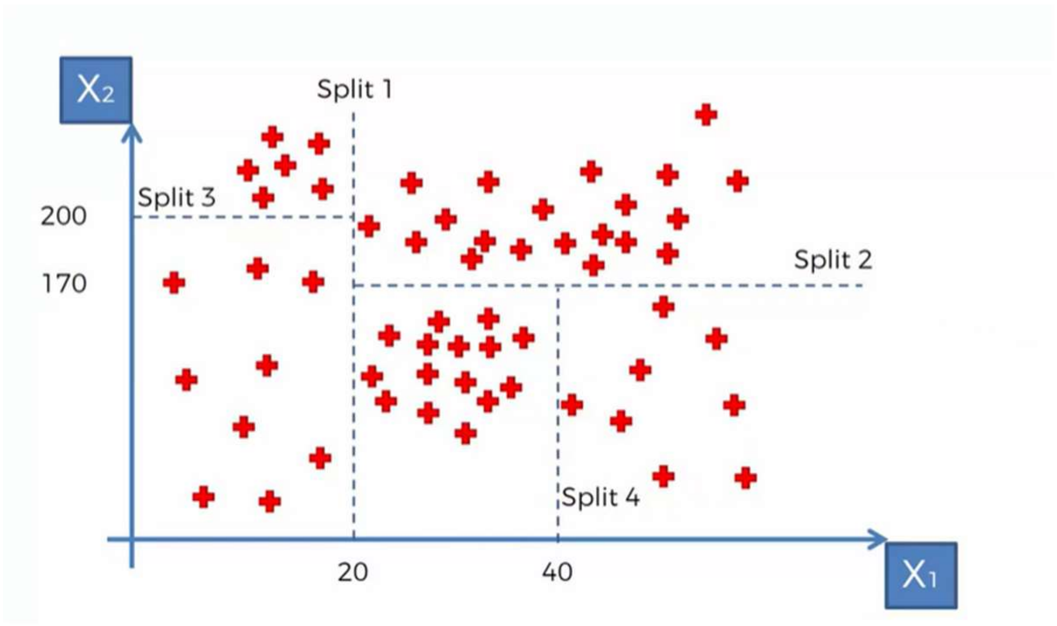
If we wanted to try an scatterplot, it would be very hard because the plots are two-dimensional, so we would see something like this.





Regression Tree

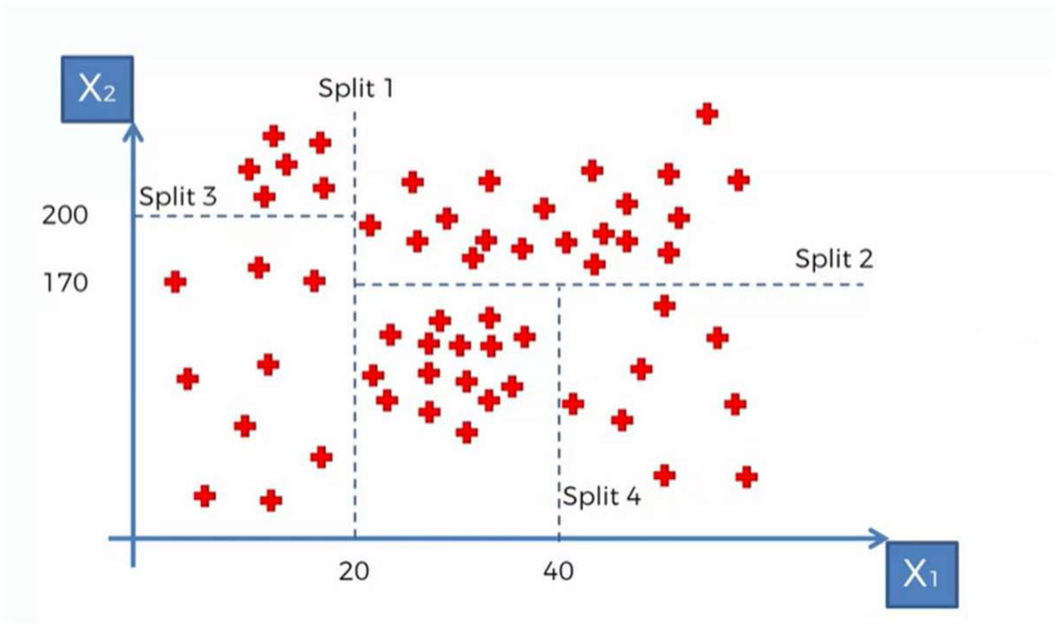
Now, knowing that in a three-dimensional plot we would see that differently, in this case, in a two-dimensional plot, we will see our data split. Imagine the following scenario.



We could find all these split using an algorithm, and this algorithm is the decision tree. It is important to understand that the split will increase the information that we have about the datapoints.

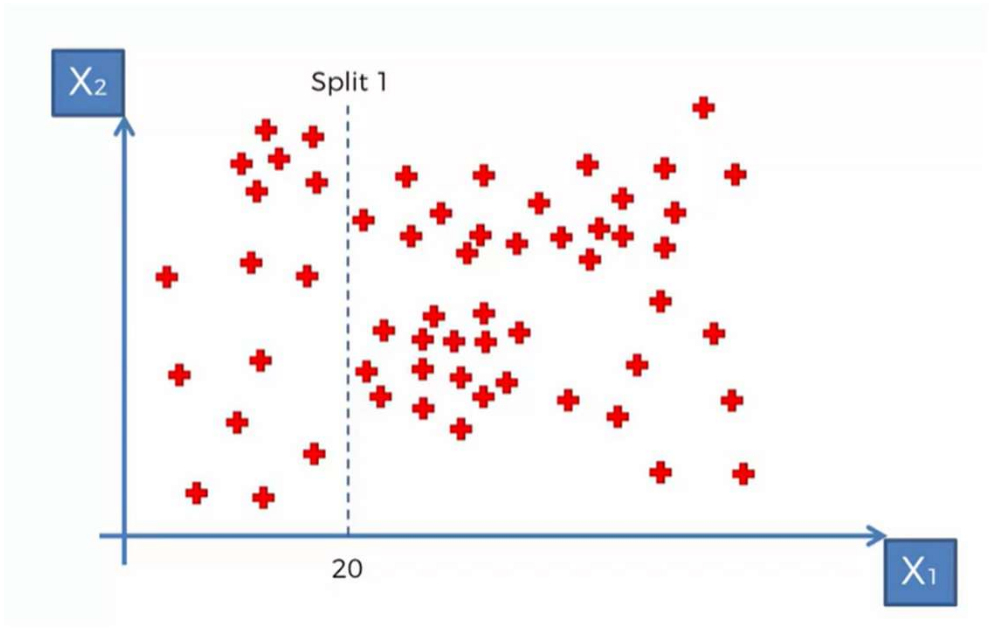
Regression Tree

In the decision tree, each split is called “leave” If we have 4 splits, we have 4 leaves in the decision tree. What we will do is using an algorithm that will find optimum splits dividing our dataset in different leaves. Let’s go a bit deeper to see how the algorithm works.

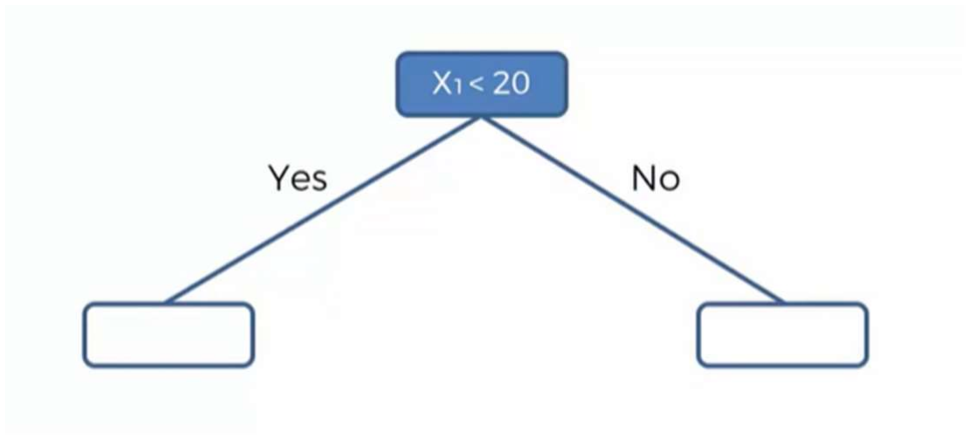


Regression Tree

The first split happened at 20.



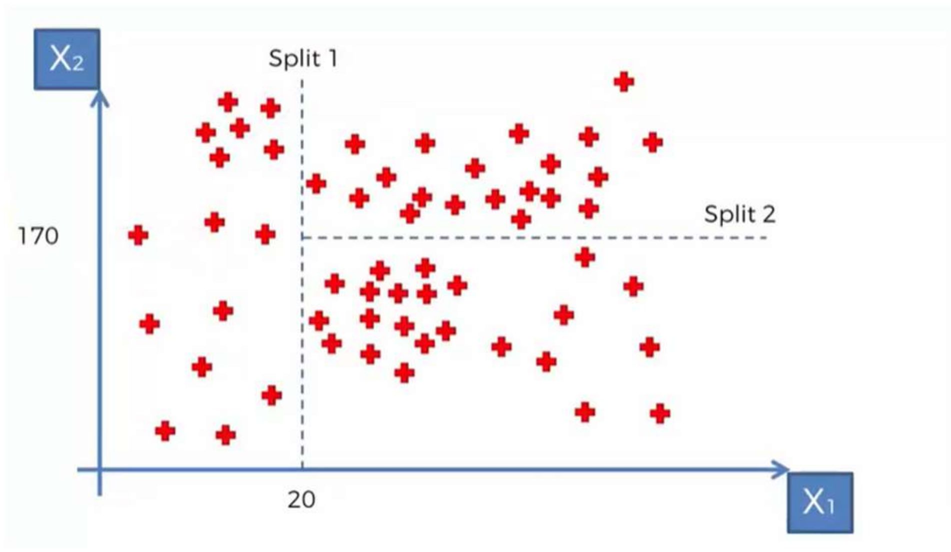
For the values up to 20 we will find 2 possible decisions

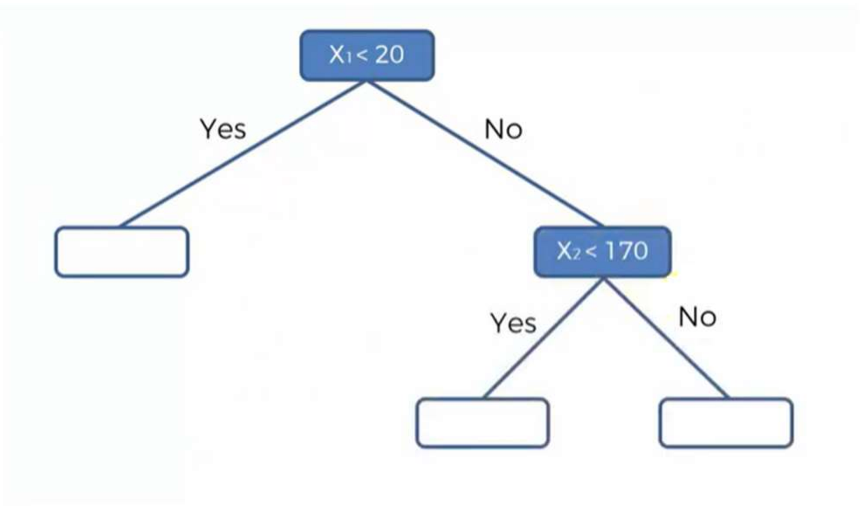


Regression Tree

Split 2 happens at 170 and only happens for the points that are greater than 20.

We answered “NO” to X1<20, and then we will have the option to decide again whether we move on for YES or NO

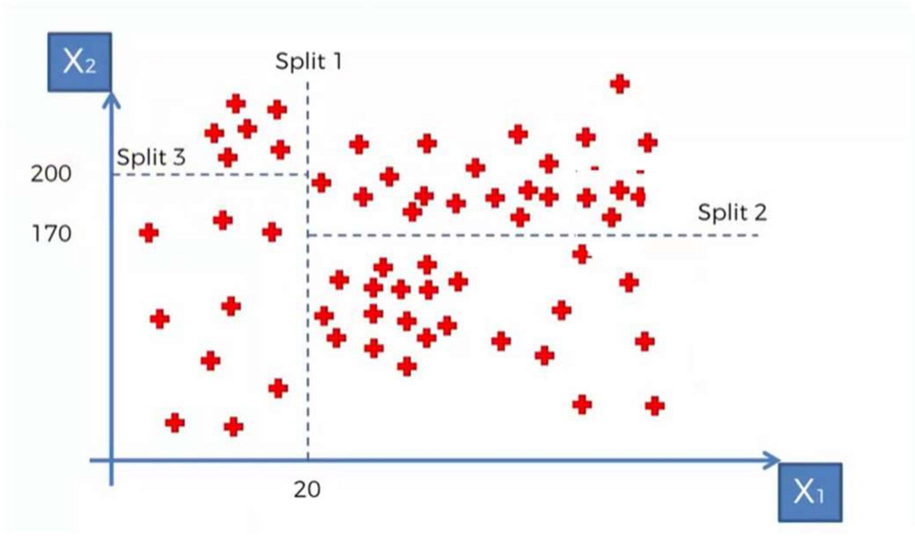


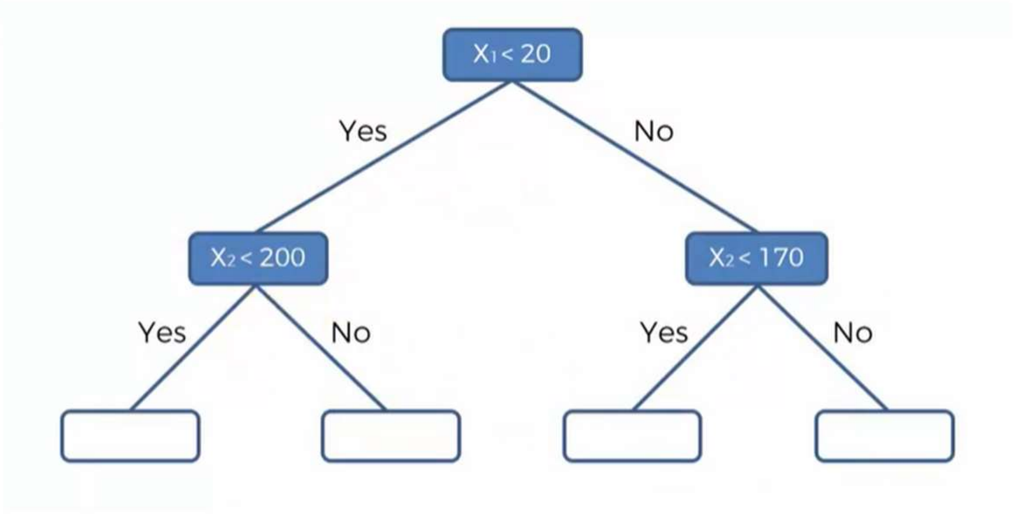


Regression Tree

Split 3 happens at 200

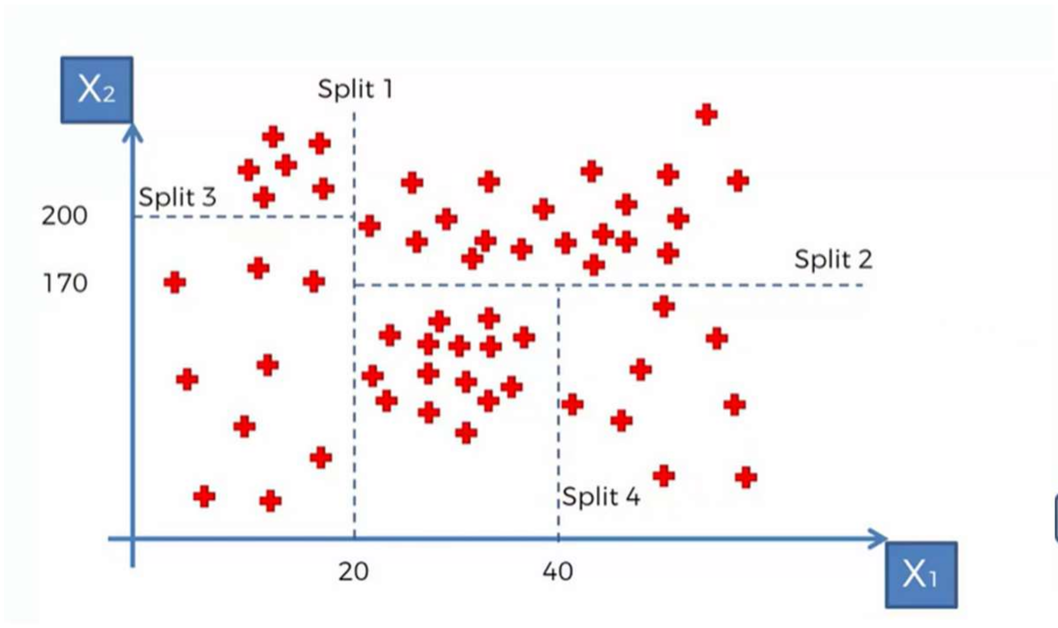
Here we check if X2 is less than 200



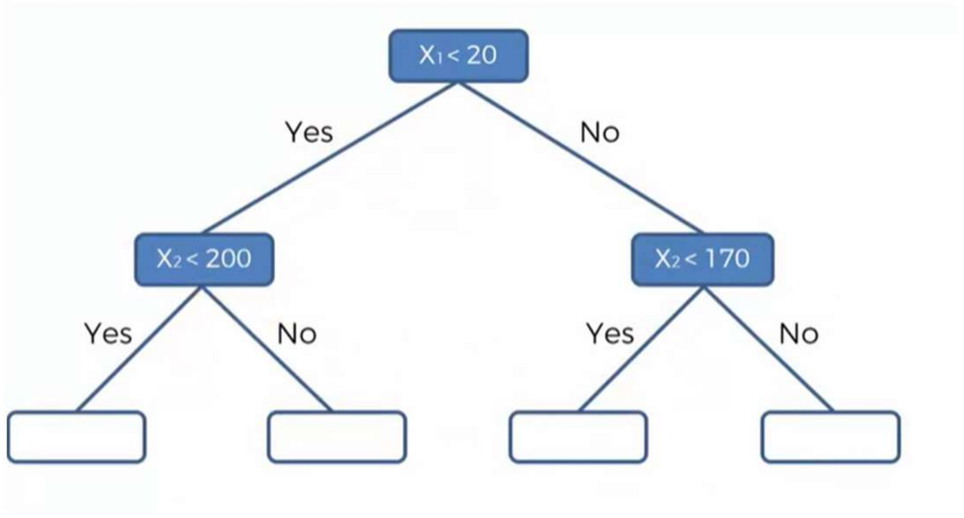


Regression Tree

Split 4 happens at 40

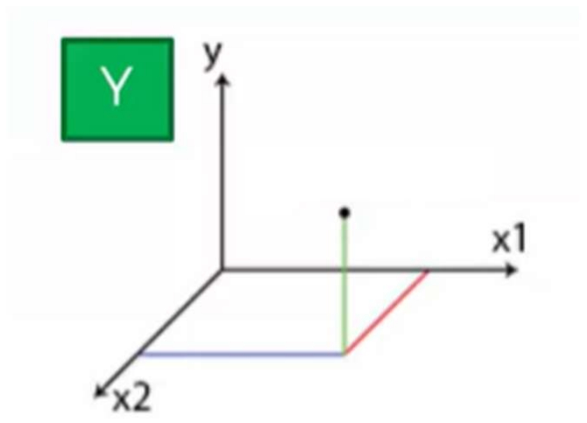


Here the question is if X1 <20, which in this case is NO and if X2 is less than 170, which in this case is YES.



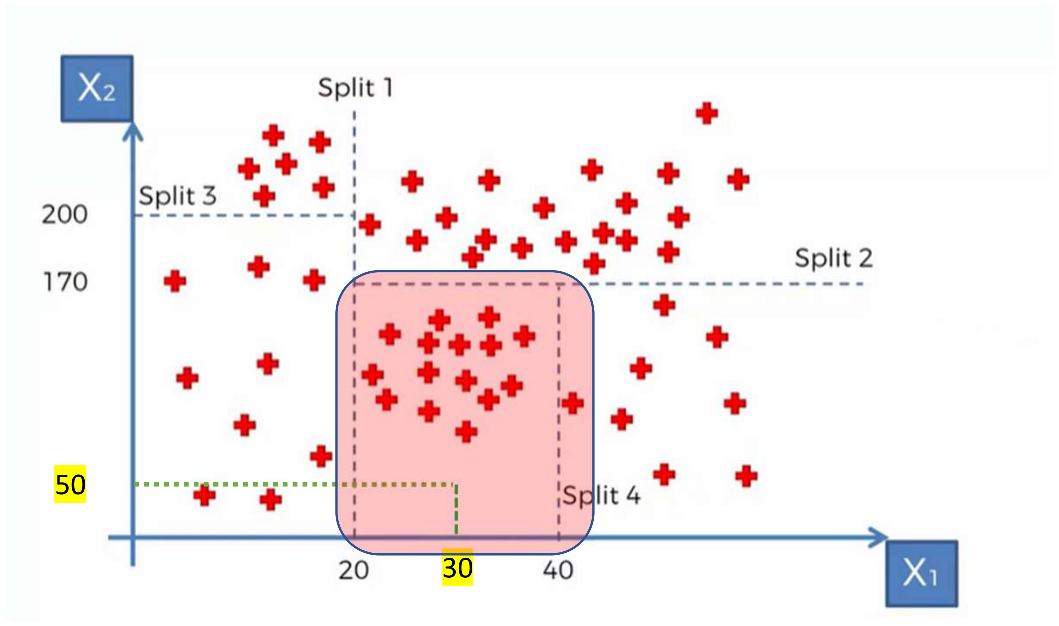
Regression Tree

Now that we understand the paths of our decision tree, how do we do to see this information in a chart? Here we need to remember our three-dimensional graph, because we need to predict what is the value that “y” will take under certain circumstances.



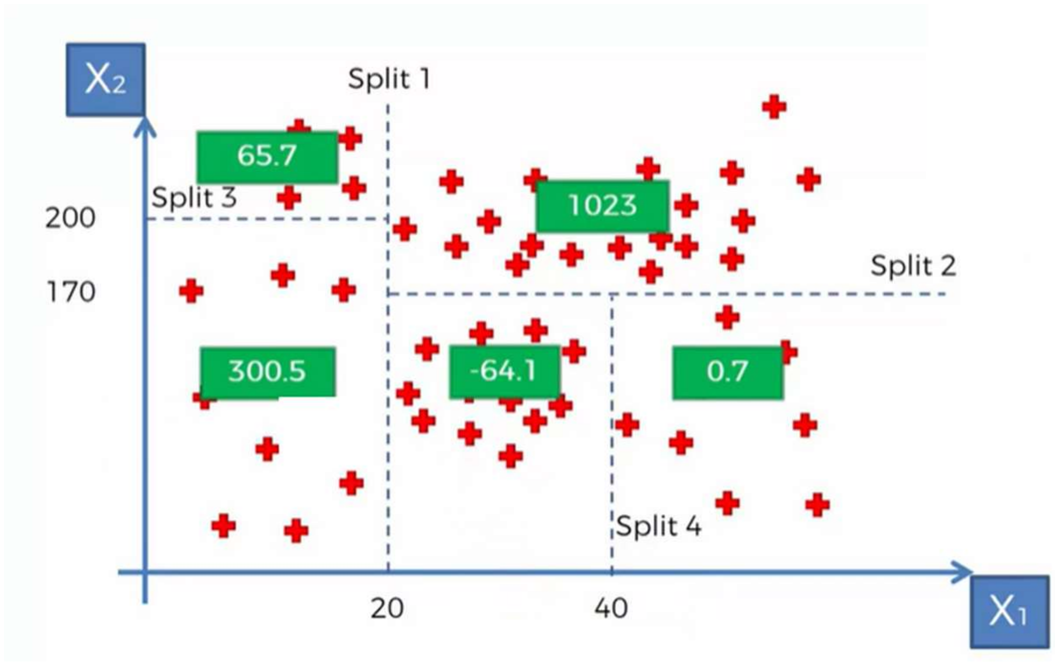
Regression Tree

Imagine that we have an observation that has X1 = 30 and X2 = 50. It would fall under the leave highlighted in the chart.



Regression Tree

The algorithm will calculate an average of “y” for each of the terminal leaves regardless of the exact point where the observation fell.



In this case, for our datapoint, the decision tree will predict y = -64.1

To sum up, the main goal is adding more information to the datapoints to better predict “y”.

Regression Tree

Some important observations.

To work on this model, there are some steps that we can skip. It happens because the algorithm will perfectly split the data, and there are no equations behind that could affect the scale, therefore we do not need to use the Feature Scaling.

Dividing the set into Training and Test is also optional for this model.

Regression Tree

Let’s try it in Python!

THAT'S ALL FOR TODAY

THANK YOU