Chapter 3 : part 6

**Decision Tree Classification**

**3.6.1 Decision Tree Regression Algorithm**

The goal of the *Decision Tree Regression Algorithm* is to *split* *data* into *similar groups*. The algorithm uses something called ***Information*** ***Entropy***, which is a mathematical process (that is super complicated).

* The point of the algorithm is to ***split the data in such a way that information is added*** and stops splitting the data when it’s unable to add any more information to the dataset.
* In the Regression Chapter, we saw the ‘Decision Tree Regression’. Regression Trees *predict* *data* of *real* *numbers*.
* Classification trees helps us to classify our data, among the categorical variables such as male or female apple or orange or different types of colors and variables of that sort.
* Whereas Regression trees are designed to help you predict outcomes which can be real numbers so for instance the salary of a person or the temperature that's going to be outside.
* In Classification trees, we *classify* *the* *data* and *predict* *categories* and. Here in this part we will be looking at but the approach is almost identical to that of the Decision Tree Regression.

|  |  |
| --- | --- |
| * Here we see a graph with *splits across different categories* using the *Decision* *Tree* *Algorithm*. * The decision tree makes splits in a way that would ***maximize the number of points in each class*** and tries to minimize entropy. So in plain English, ***it groups as many of the green and red dots together***. | Decision Tree Splits |
| * The decision tree would like this: * This is the method that will be used to add a new data point and classify it to the graph. * Additionally, we *don’t always* have to go all the way to the *very end of the tree*. * Most of the time the tree will be very long and so at a *reasonable point* you *could* *stop* and have it *run* *a* *probability* of where to *classify* *the* *point*. | Decision Tree Splits |

* For example, let’s say we stop going through the graph at . If the statement is True then we would have, in this case a 100% chance of it being red. Otherwise, ***instead going further down the list***, we could say that since there are more Green Dots than Red Ones, I could make a prediction and say that the point will be green. This is extremely useful, as it is a lot *less computationally* expensive.
* Decision Tree’s on their own is not a very powerful tool. However, with the help of other tools like Random Forest Trees and Gradient Boosting it has become much more powerful in the past few years.

|  |  |
| --- | --- |
| * How it works: Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. The algorithm is just split the data so that point of a class will be maximum.      * The algorithms going to find optimal splits that are going to maximize the number of different points in each one of leaves (leaves are splitted data pockets, final leaves are actually called a terminal leaves). * For example our data-set is splitted as: * For more than two feature variables, the Tree just grow bigger. |  |

* It is a tree-structured classifier, where ***internal*** ***nodes*** represent the ***features*** of a dataset.
* ***Branches*** represent the ***decision*** rules and
* Each ***leaf*** ***node*** represents the ***outcome***.
* Decision Node and Leaf Node: In a ***Decision*** ***tree***, there are two nodes, which are the ***Decision*** ***Node*** and ***Leaf*** ***Node***.
* *Decision nodes* are used to make any *decision* and have *multiple* *branches*, whereas
* *Leaf nodes* are the *output* of those *decisions* and ***do not contain any further branches***. The decisions or the test are performed on the basis of features of the given dataset.
* It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
* It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
* In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm.
* A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees. Below diagram explains the general structure of a decision tree:



Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.

**3.6.2 Implementation of Decision Tree in Python**

**from** sklearn**.**tree **import** DecisionTreeClassifier

classifier = **DecisionTreeClassifier**(criterion = 'entropy', random\_sate = 0)

classifier**.fit**(X\_train, y\_train)

* No need scaled feature: Decision Tree is not based on Euclidean distance, so we need not really apply feature scaling here. Feature scaling is important when your algorithm is based on Euclidean distance.
* However in we are building some trees, so we want to plot those trees in real values rather than scaled values.
* Why we are keeping scaled feature: But we used a "high resolution" plot to "visualize the result". It generates faster if the features are scaled, so in this case we are keeping our scaled feature.
* If you want to plot some decision trees like the real tree itself then you can remove this feature scaling part here to have interpretation of your valuables.
* Parameters:

**criterion** : {"**gini**", "**entropy**"}, default="gini"

* The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
* Most useful classifier and most common decision tree is based on "Entropy", eg: "Maximum Entropy" for NLP.
* So that the *final nodes* of your tree to be as much *homogeneous* as *possible*. I.e. after each split the *more homogeneous* is a *group of users*, the more the *entropy* is *reduced* from the *parent node* to the *group child node*.
* So after the *split* if the *entropy* in the *resulting child node* is *zero* then that means that this *child* *node* is a *fully* *homogeneous* *group of users* (only users of the same class) and the information gain mentioned here is basically the difference between the entropies before and after displays.

***Practiced version***

#*----------- Naive Bayes model ----------*

#*Library*

**import** pandas **as** pd

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

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dataSet**.**iloc[:, [2,3]]**.**values

y = dataSet**.**iloc[:, 4]**.**values

        #*Feature-Scaling after Data Split*

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

#*Feature-Scaling*

**from** sklearn**.**preprocessing **import** StandardScaler

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)

#*Fit train set to Naïve Bayes classifier: No parmeter is needed*

**from** sklearn**.**tree **import** DecisionTreeClassifier

clsFier = **DecisionTreeClassifier**(criterion="entropy", random\_state=0)

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualising the Training set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('Decision Tree (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**from** matplotlib**.**colors **import** ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np**.meshgrid**(np**.arange**(start = X\_set[:, 0]**.min**() - 1, stop = X\_set[:, 0]**.max**() + 1, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             alpha = 0.5, cmap = **ListedColormap**(('red', 'green')))

pLt**.xlim**(X1**.min**(), X1**.max**())

pLt**.ylim**(X2**.min**(), X2**.max**())

**for** i, j **in** **enumerate**(np**.unique**(y\_set)):

    pLt**.scatter**(X\_set[y\_set **==** j, 0], X\_set[y\_set **==** j, 1],

                c = **ListedColormap**(('red', 'green'))(i), label = j)

pLt**.title**('Decision Tree (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*python prctc\_DTC.py*

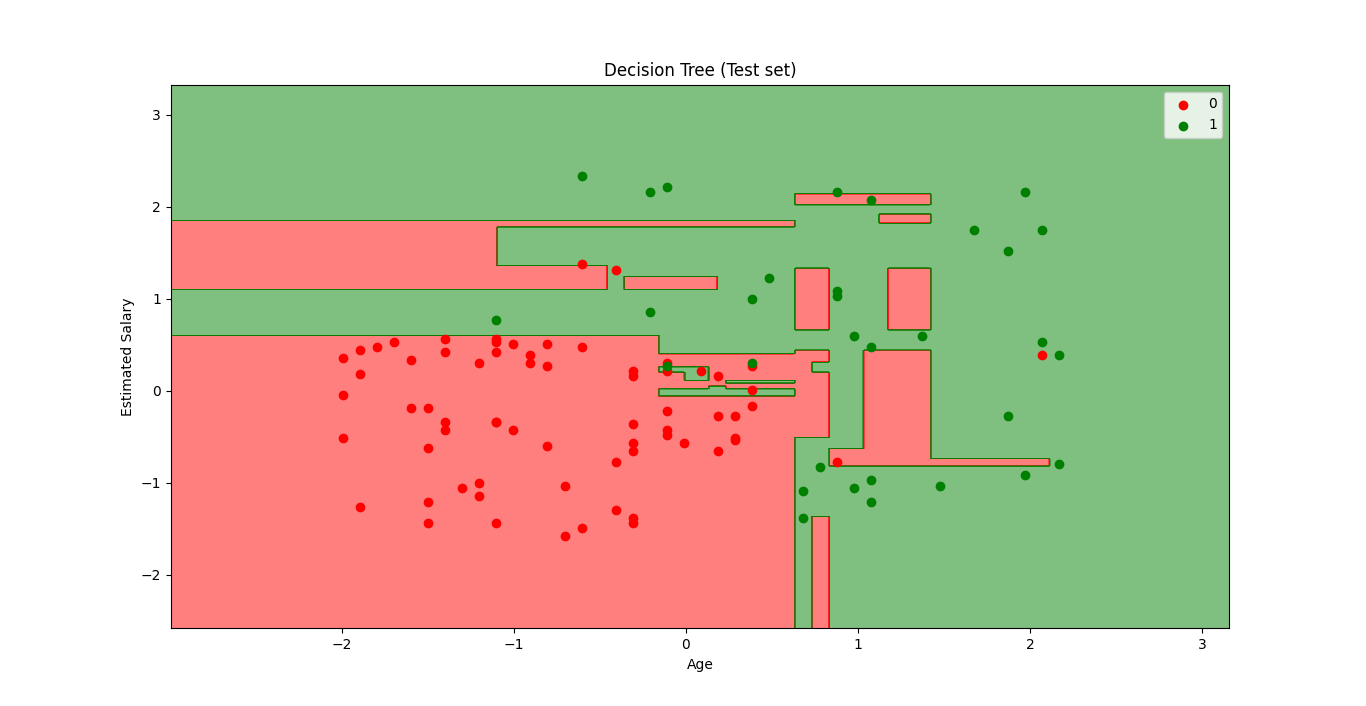
Test Data vs Predicted data and CONFUSION MATRIX

|  |  |
| --- | --- |
|  |  |

Training set Plot



Test set Plot



* From above figures, we see that, *Prediction boundary* is composed on Horizontal and vertical lines. This model actually considered to classify all red and green point at 100%.
* This model is Kind of Over-fitted: This kind of like over-fitting, because you see it's trying to catch every single user into the right category.