Decision Tree

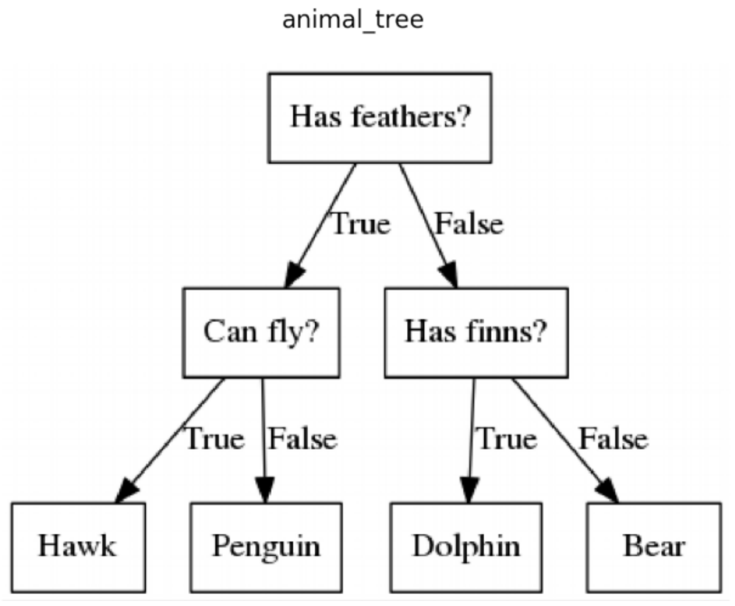
- Decision tree is one of the most popular machine learning algorithms used all along.
- Decision trees are used for both classification and regression problems.

1. Why Decision trees?

We have couple of other algorithms there, so why do we have to choose Decision trees??

- Decision tress often mimic the human level thinking so its so simple to understand the data and make some good interpretations.
- Decision trees actually make you see the logic for the data to interpret(not like black box algorithms like SVM,NN,etc..)
- A decision tree follows a set of if-else conditions to visualize the data and classify it according to the conditions.

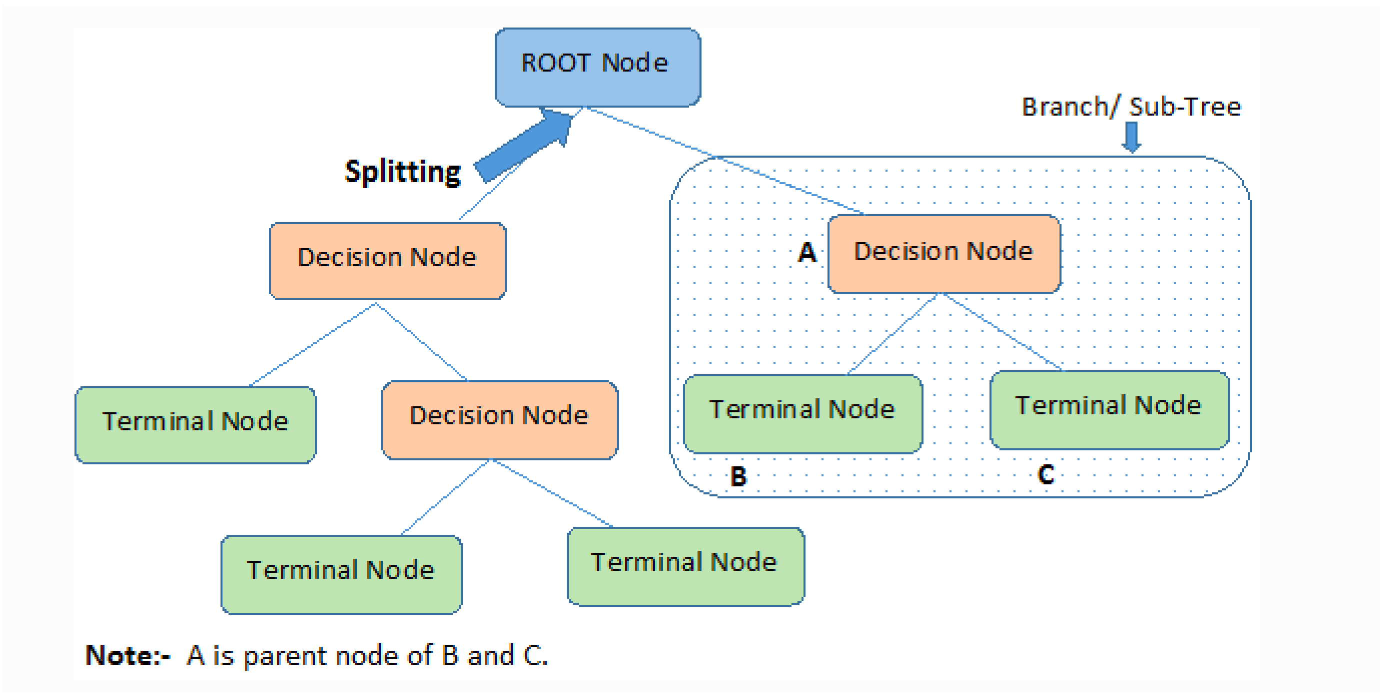
For example :



- A decision tree is a tree where each node represents a feature(attribute), each link(branch) represents a decision(rule) and each leaf represents an outcome(categorical or continues value).

Important terminology In DT

- Root Node: This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
- Branch or Sub-Tree: A part of the entire decision tree is called branch or sub-tree.
- Splitting: Dividing a node into two or more sub-nodes based on if-else conditions.
- Decision Node: After splitting the sub-nodes into further sub-nodes, then it is called as the decision node.
- Leaf or Terminal Node: This is the end of the decision tree where it cannot be split into further sub-nodes.
- Pruning: Removing a sub-node from the tree is called pruning.



Types of Decision Trees

- Types of decision tree is based on the type of target variable we have. It can be of two types:

1. Categorical Variable Decision Tree: Decision Tree which has categorical target variable then it called as categorical variable decision tree. Example:- In above scenario of student problem, where the target variable was “Student will play cricket or not” i.e. YES or NO.

2. Continuous Variable Decision Tree: Decision Tree has continuous target variable then it is called as Continuous Variable Decision Tree.

Three Important Term In DT

- Entropy(H)
- Information Gain(IG)
- Gini Impurity

Entropy

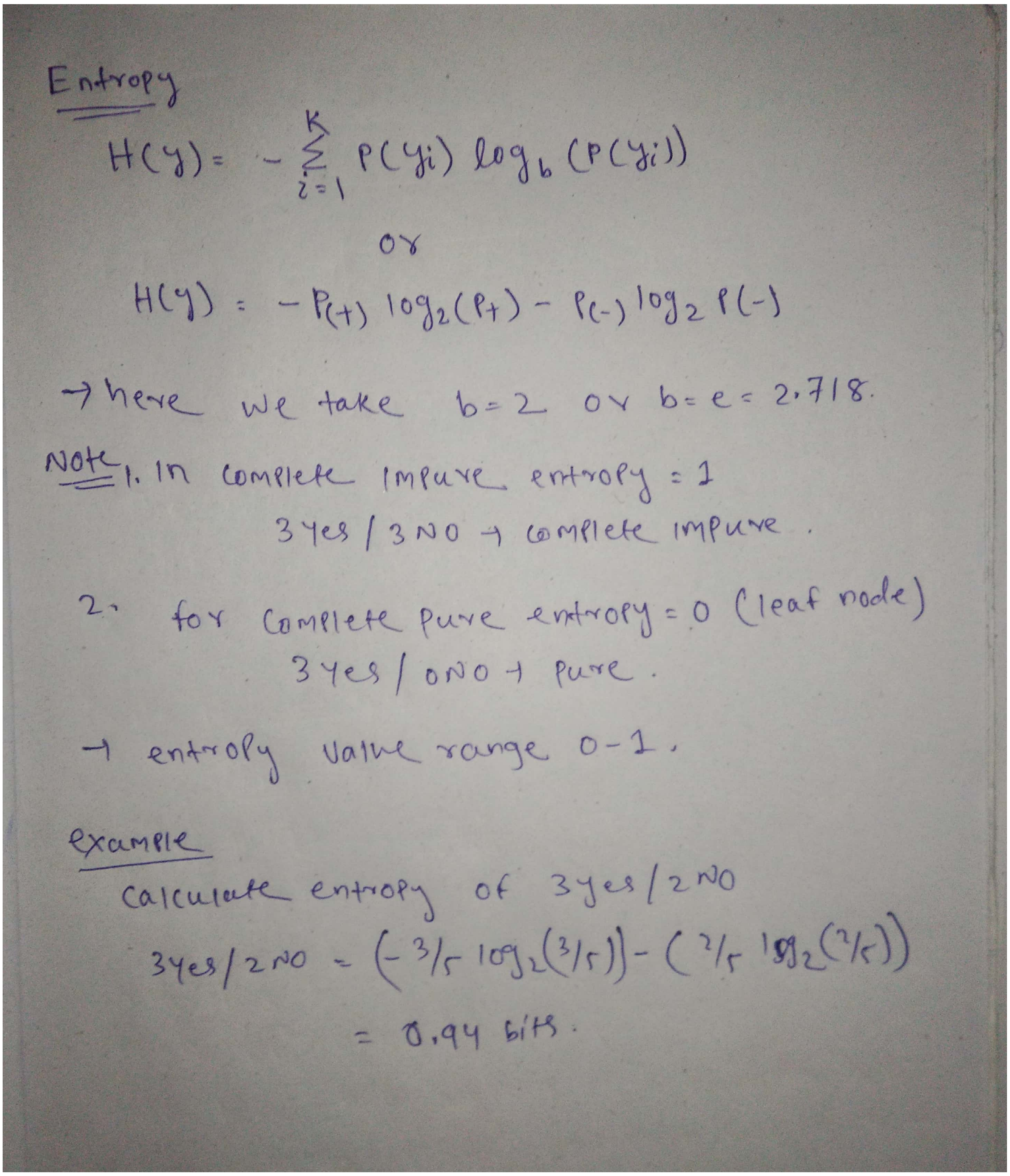
-> It is select the best feature/attribute in DT for Splitting Purpose

-> It helps to measure purity of split

-> Pure split is either get yes or no but not both.

-> Eg. 3yes/2No - it is not pure But 3Yes/0No - it is pure

-> we keep split whenever pure split not come.



Entropy

$$H(y) = -\sum_{i=1}^{K} P(y_i) \log_b (P(y_i))$$

or

$$H(y) = -P_{(+)} \log_2(P_+) - P_{(-)} \log_2 P(-)$$

→ here we take $b = 2$ or $b = e = 2.718$.

Note 1. In complete impure entropy $= 1$

3 yes / 3 NO → complete impure.

2. for complete pure entropy $= 0$ (leaf node)

3 yes / 0 NO → pure.

→ entropy value range 0-1.

example

calculate entropy of 3 yes / 2 NO

$$3\text{yes}/2\text{NO} = \left(-\tfrac{3}{5} \log_2\left(\tfrac{3}{5}\right)\right) - \left(\tfrac{2}{5} \log_2\left(\tfrac{2}{5}\right)\right)$$

$$= 0.94 \text{ bits.}$$

2. Information Gain(IG)

-> Information gain is calculated by comparing the entropy of the dataset before and after a transformation.

-> If there are many entropy then we compute average of all entropy and compare to find which is best this done by information gain

-> Information gain is the reduction in entropy or surprise by transforming a dataset and is often used in training decision trees.

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

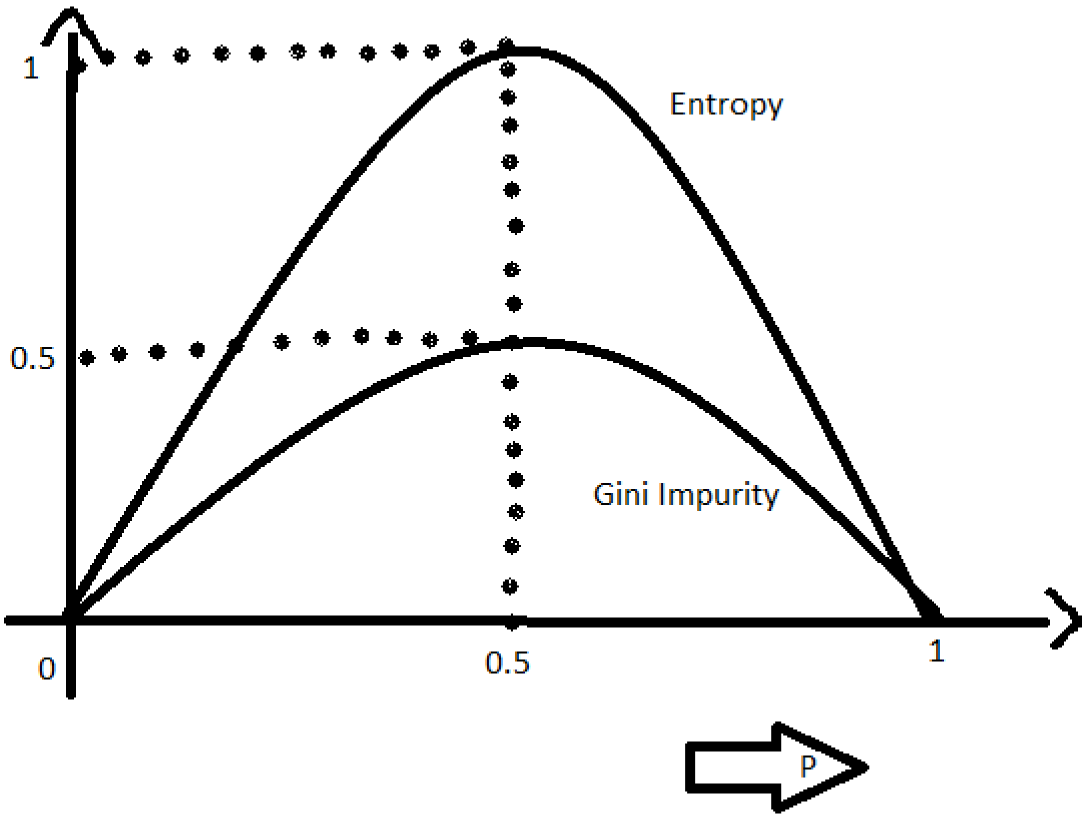
3. Gini Impurity

-> Both Entropy and Gini Impurity Calculate the purity of split in DT.

-> But Gini Impurity is better than Entropy. So we use Gini Impurity to find purity of split.

-> Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.

Lets plot the Entropy Vs Gini impurity



$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

$$Gini(E) = 1 - \textstyle\sum_{j=1}^{c} p_j^2$$

-> Entropy Max Value is 1 but Gini Impurity Max Value is 0.5.

-> As Gini Impurity has no logarithmic function so it is more efficient than Entropy.

Tree Pruning

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.

Post-pruning

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is reduction in accuracy then the node not be expanded and should be converted in a leaf node.

Pre-pruning

Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as the tree is generated.

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM). Here it use Entropy or Gini Impurity.

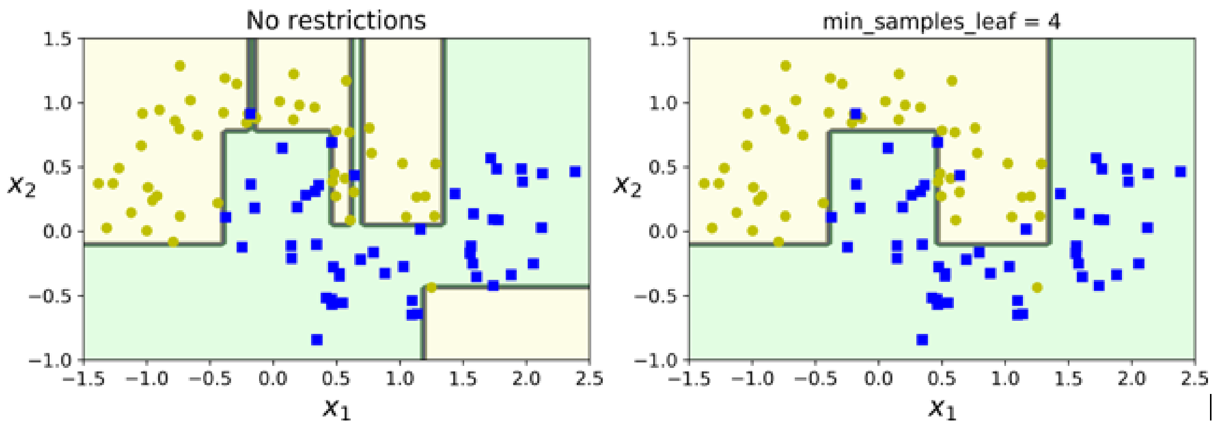
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

See An Example Of Above Algorithm

- Here YOu see the overview of DT, On the left, the Decision Tree is trained with the default hyperparameters (i.e., no restrictions), and on the right the Decision Tree is trained with min_sam ples_leaf=4. It is quite obvious that the model on the left is overfitting, and the model on the right will probably generalize better.



Lets Implement With Python

Importing the libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [3]:

```
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 =
```

Feature Scaling

In [4]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Decision Tree Classification model on the Training

set

In [5]:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[5]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting the Test set results

In [6]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

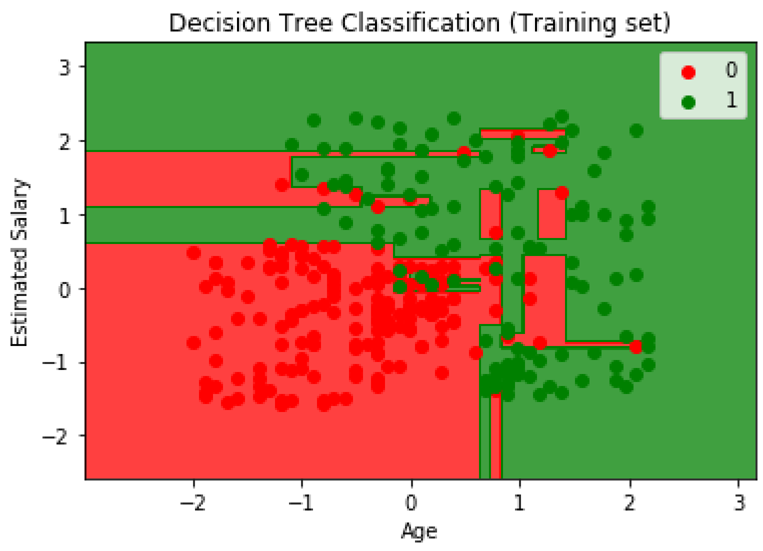
```
[[62  6]
 [ 3 29]]
```

Visualising the Training set results

In [8]:

```python
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,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, 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 Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



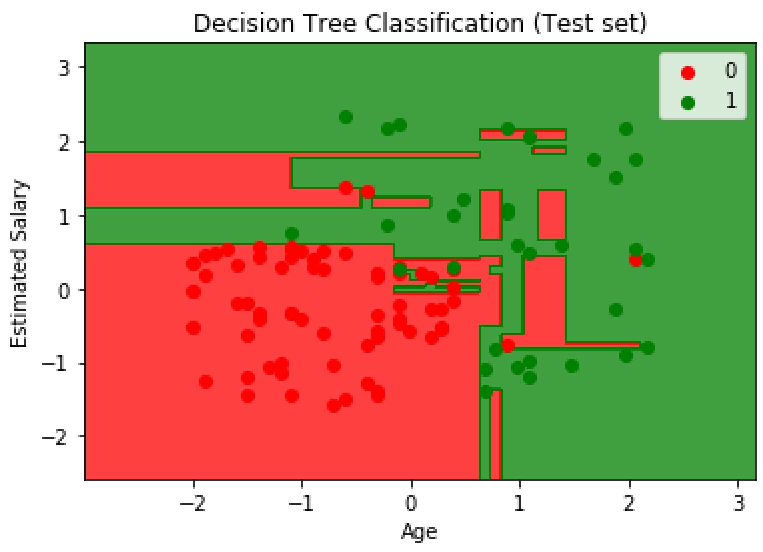
- The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.
- As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

Visualising the Test set results

In [9]:

```
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,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, 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 Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



As we can see in the above image that there are some green data points within the red region and vice versa. So, these are the incorrect predictions.