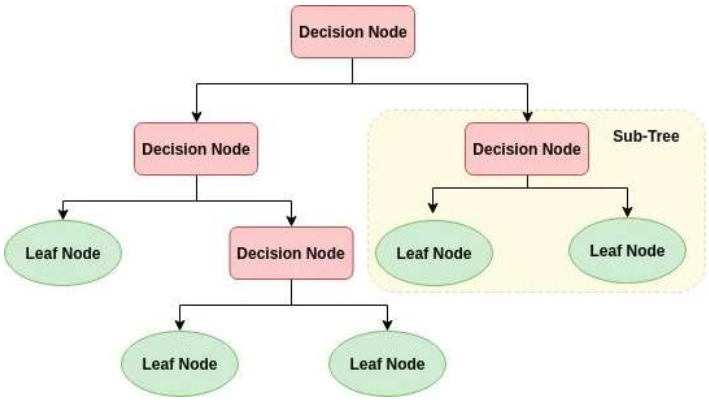
Decision-Tree Classifier Tutorial

Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

Classification and Regression Trees (CART)

Nowadays, Decision Tree algorithm is known by its modern name CART which stands for Classification and Regression Trees. Classification and Regression Trees or CART is a term introduced by Leo Breiman to refer to Decision Tree algorithms that can be used for classification and regression modeling problems.

The CART algorithm provides a foundation for other important algorithms like bagged decision trees, random forest and boosted decision trees. In this kernel, I will solve a classification problem. So, I will refer the algorithm also as Decision Tree Classification problem.

Decision Tree algorithm terminology

- In a Decision Tree algorithm, there is a tree like structure in which each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. The paths from the root node to leaf node represent classification rules.
- We can see that there is some terminology involved in Decision Tree algorithm. The terms involved in Decision Tree algorithm are as follows:

- Root Node

It represents the entire population or sample. This further gets divided into two or more homogeneous sets.

- Splitting

It is a process of dividing a node into two or more sub-nodes.

- Decision Node

When a sub-node splits into further sub-nodes, then it is called a decision node.

- Leaf/Terminal Node

Nodes that do not split are called Leaf or Terminal nodes.

- Pruning

When we remove sub-nodes of a decision node, this process is called pruning. It is the opposite process of splitting.

- Branch/Sub-Tree

A sub-section of an entire tree is called a branch or sub-tree.

- Parent and Child Node

A node, which is divided into sub-nodes is called the parent node of sub-nodes where sub-nodes are the children of a parent node.

How does the Decision Tree Algorithm Work?

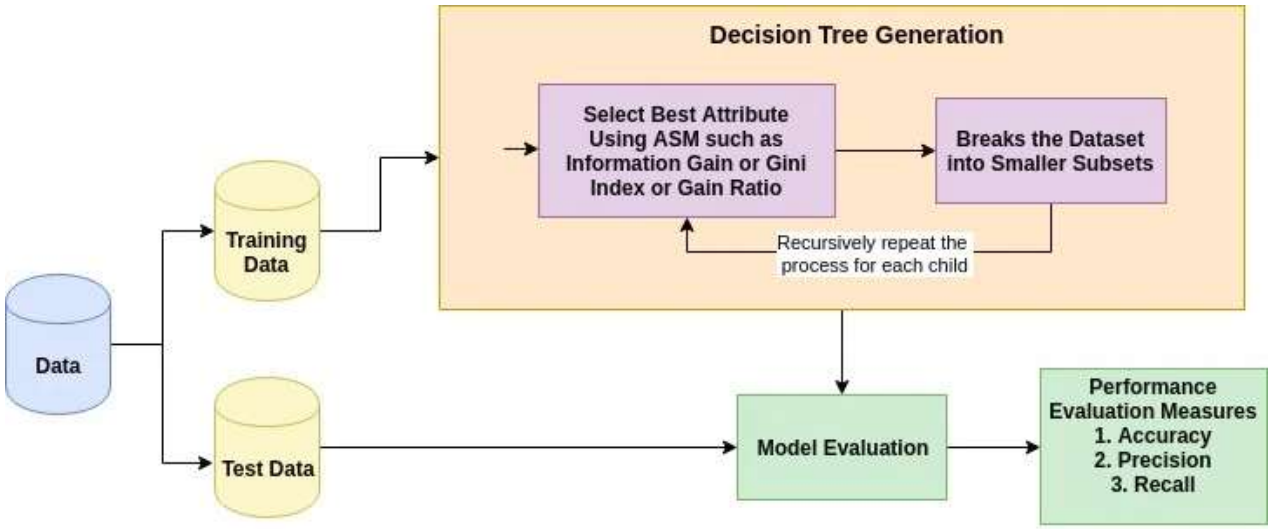
The basic idea behind any decision tree algorithm is as follows:

1- Select the best attribute using Attribute Selection Measures(ASM) to split the records.

2- Make that attribute a decision node and breaks the dataset into smaller subsets.

3- Starts tree building by repeating this process recursively for each child until one of the condition will match:

- All the tuples belong to the same attribute value.
- There are no more remaining attributes.
- There are no more instances.



Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature(or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute (Source). In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy referred as the randomness or the impurity in the system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \text{ X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

Where,

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- |Dj|/|D| acts as the weight of the jth partition.
- InfoA(D) is the expected informa-tion required to classify a tuple from D based on the partitioning by A.
- The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N(). ### Gain Ratio

Information gain is biased for the attribute with many outcomes. It means it prefers the attribute with a large number of distinct values. For instance, consider an attribute with a unique identifier such as customer_ID has zero info(D) because of pure partition. This maximizes the information gain and creates useless partitioning.

C4.5, an improvement of ID3, uses an extension to information gain known as the gain ratio. Gain ratio handles the issue of bias by normalizing the information gain using Split Info. Java implementation of the C4.5 algorithm is known as J48, which is available in WEKA data mining tool.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

Where,

- |Dj|/|D| acts as the weight of the jth partition.
- v is the number of discrete values in attribute A.

The gain ratio can be defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The attribute with the highest gain ratio is chosen as the splitting attribute

Gini Index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points.

$$\text{Gini(D)}=1 - \sum_{i=1}^{m} \text{Pi}^2$$

Where, pi is the probability that a tuple in D belongs to class Ci.

The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

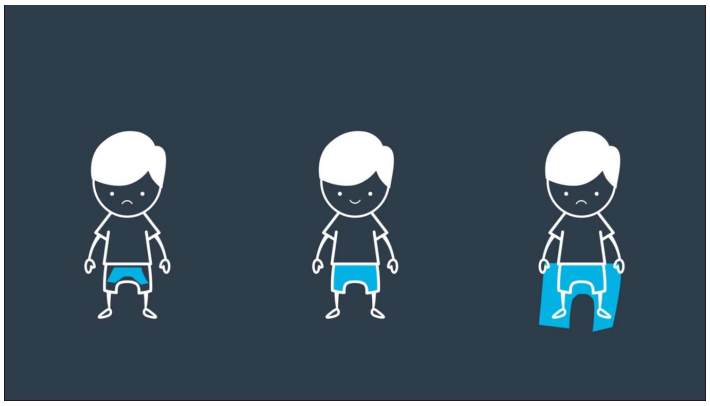
$$\text{Gini}_A(D)=\frac{|D1|}{|D|}\text{Gini}(D_1)+\frac{|D2|}{|D|}\text{Gini}(D_2)$$

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with minimum Gini index is chosen as the splitting attribute.

Overfitting in Decision Tree algorithm



Overfitting is a practical problem while building a Decision-Tree model. The problem of overfitting is considered when the algorithm continues to go deeper and deeper to reduce the training-set error but results with an increased test-set error. So, accuracy of prediction for our model goes down. It generally happens when we build many branches due to outliers and irregularities in data.

Two approaches which can be used to avoid overfitting are as follows:

- Pre-Pruning
- Post-Pruning

Pre-Pruning

In pre-pruning, we stop the tree construction a bit early. We prefer not to split a node if its goodness measure is below a threshold value. But it is difficult to choose an appropriate stopping point.

Post-Pruning

In post-pruning, we go deeper and deeper in the tree to build a complete tree. If the tree shows the overfitting problem then pruning is done as a post-pruning step. We use the cross-validation data to check the effect of our pruning. Using cross-validation data, we test whether expanding a node will result in improve or not. If it shows an improvement, then we can continue by expanding that node. But if it shows a reduction in accuracy then it should not be expanded. So, the node should be converted to a leaf node.

Pros VS. Cons

Pros

- Decision trees are easy to interpret and visualize.
- It can easily capture Non-linear patterns.
- It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.
- It can be used for feature engineering such as predicting missing values, suitable for variable selection.
- The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.

Cons

- Sensitive to noisy data. It can overfit noisy data.
- The small variation(or variance) in data can result in the different decision tree. This can be reduced by bagging and boosting algorithms.
- Decision trees are biased with imbalance dataset, so it is recommended that balance out the dataset before creating the decision tree.

Decision Tree Classifier Building in Scikit-learn :

Importing the libraries

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

Importing the dataset

In [2]:
```
df = pd.read_csv(r"../input/social-network-ads/Social_Network_Ads.csv")
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [3]:
```
type(df)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

In [4]:
```
df.head(3)
```

Out[4]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0

In [5]:
```
df.shape
```

Out[5]:
```
(400, 3)
```

In [6]:
```
df
```

Out[6]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

In [7]:
```
y
```

Out[7]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1])
```

Splitting the dataset into the Training set and Test set

In [8]:
```
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, ra
ndom_state = 0)
```

Feature Scaling

In [9]:
```
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 [10]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[10]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Predicting a new result

In [11]:
```
print(classifier.predict(sc.transform([[40,33000]])))
```

```
[0]
```

In [12]:
```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [13]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

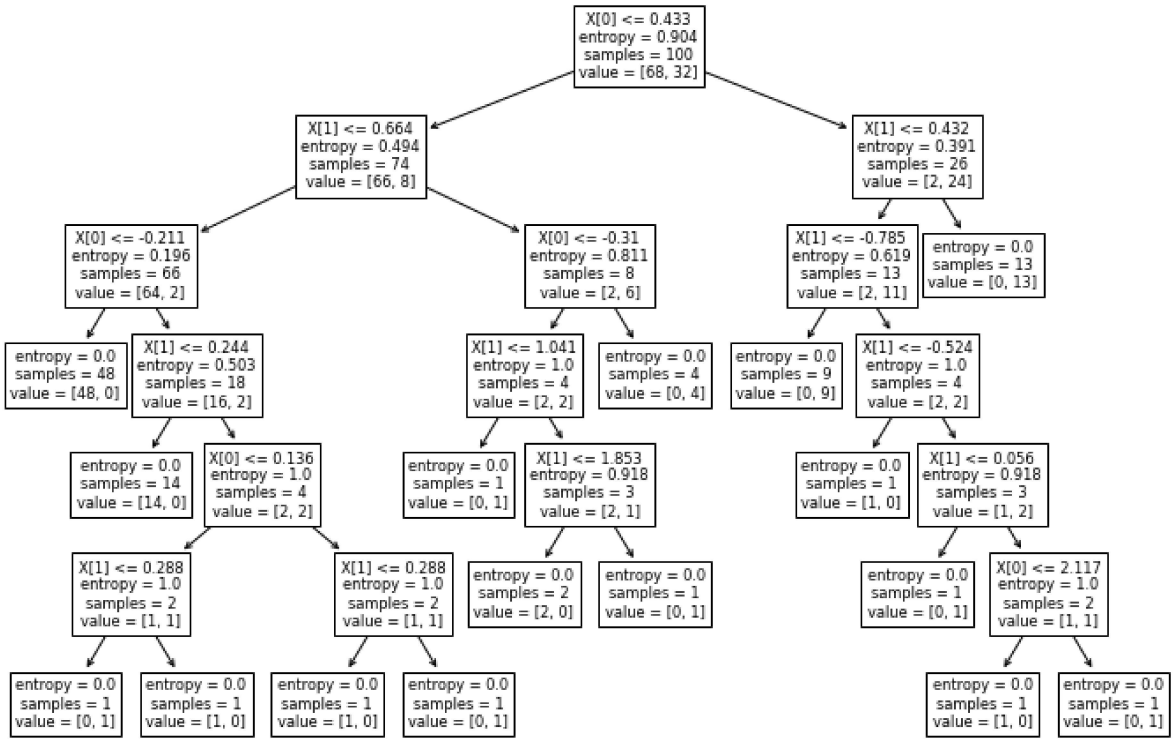
```
[[62  6]
 [ 3 29]]
```

Out[13]:
```
0.91
```

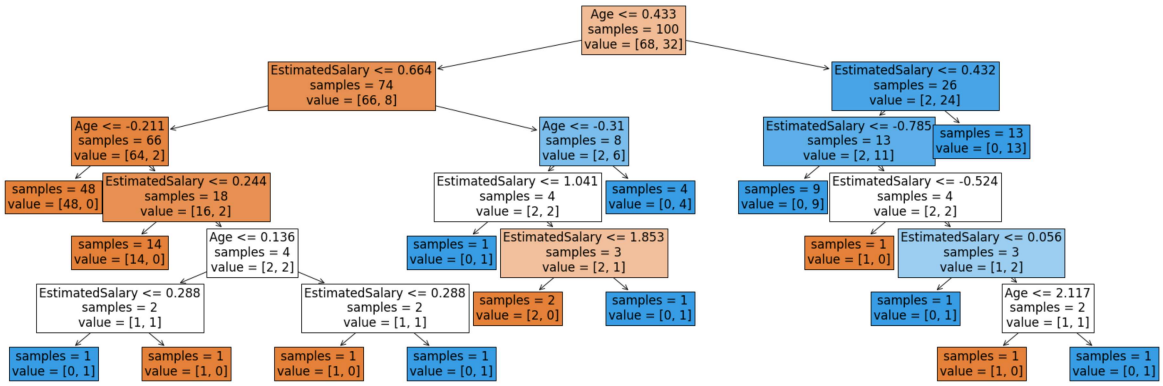
In [14]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(classifier.fit(X_test, y_test))
```

Out[14]:
```
[Text(0.5416666666666666, 0.9285714285714286, 'X[0] <= 0.433\nentropy = 0.904
\nsamples = 100\nvalue = [68, 32]'),
 Text(0.3055555555555556, 0.7857142857142857, 'X[1] <= 0.664\nentropy = 0.494
\nsamples = 74\nvalue = [66, 8]'),
 Text(0.1111111111111111, 0.6428571428571429, 'X[0] <= -0.211\nentropy = 0.19
6\nsamples = 66\nvalue = [64, 2]'),
 Text(0.05555555555555555, 0.5, 'entropy = 0.0\nsamples = 48\nvalue = [48,
0]'),
 Text(0.16666666666666666, 0.5, 'X[1] <= 0.244\nentropy = 0.503\nsamples = 18
\nvalue = [16, 2]'),
 Text(0.1111111111111111, 0.35714285714285715, 'entropy = 0.0\nsamples = 14\n
value = [14, 0]'),
 Text(0.2222222222222222, 0.35714285714285715, 'X[0] <= 0.136\nentropy = 1.0
\nsamples = 4\nvalue = [2, 2]'),
 Text(0.1111111111111111, 0.21428571428571427, 'X[1] <= 0.288\nentropy = 1.0
\nsamples = 2\nvalue = [1, 1]'),
 Text(0.05555555555555555, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\n
value = [0, 1]'),
 Text(0.16666666666666666, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\n
value = [1, 0]'),
 Text(0.3333333333333333, 0.21428571428571427, 'X[1] <= 0.288\nentropy = 1.0
\nsamples = 2\nvalue = [1, 1]'),
 Text(0.2777777777777778, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nv
alue = [1, 0]'),
 Text(0.3888888888888889, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nv
alue = [0, 1]'),
 Text(0.5, 0.6428571428571429, 'X[0] <= -0.31\nentropy = 0.811\nsamples = 8\n
value = [2, 6]'),
 Text(0.4444444444444444, 0.5, 'X[1] <= 1.041\nentropy = 1.0\nsamples = 4\nva
lue = [2, 2]'),
 Text(0.3888888888888889, 0.35714285714285715, 'entropy = 0.0\nsamples = 1\nv
alue = [0, 1]'),
 Text(0.5, 0.35714285714285715, 'X[1] <= 1.853\nentropy = 0.918\nsamples = 3
\nvalue = [2, 1]'),
 Text(0.4444444444444444, 0.21428571428571427, 'entropy = 0.0\nsamples = 2\nv
alue = [2, 0]'),
 Text(0.5555555555555556, 0.21428571428571427, 'entropy = 0.0\nsamples = 1\nv
alue = [0, 1]'),
 Text(0.5555555555555556, 0.5, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.7777777777777778, 0.7857142857142857, 'X[1] <= 0.432\nentropy = 0.391
\nsamples = 26\nvalue = [2, 24]'),
 Text(0.7222222222222222, 0.6428571428571429, 'X[1] <= -0.785\nentropy = 0.61
9\nsamples = 13\nvalue = [2, 11]'),
 Text(0.6666666666666666, 0.5, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.7777777777777778, 0.5, 'X[1] <= -0.524\nentropy = 1.0\nsamples = 4\nv
alue = [2, 2]'),
 Text(0.7222222222222222, 0.35714285714285715, 'entropy = 0.0\nsamples = 1\nv
alue = [1, 0]'),
 Text(0.8333333333333334, 0.35714285714285715, 'X[1] <= 0.056\nentropy = 0.91
8\nsamples = 3\nvalue = [1, 2]'),
 Text(0.7777777777777778, 0.21428571428571427, 'entropy = 0.0\nsamples = 1\nv
alue = [0, 1]'),
 Text(0.8888888888888888, 0.21428571428571427, 'X[0] <= 2.117\nentropy = 1.0
\nsamples = 2\nvalue = [1, 1]'),
 Text(0.8333333333333334, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nv
alue = [1, 0]'),
 Text(0.9444444444444444, 0.07142857142857142, 'entropy = 0.0\nsamples = 1\nv
```

```
alue = [0, 1]'),
 Text(0.8333333333333334, 0.6428571428571429, 'entropy = 0.0\nsamples = 13\nv
alue = [0, 13]')]
```

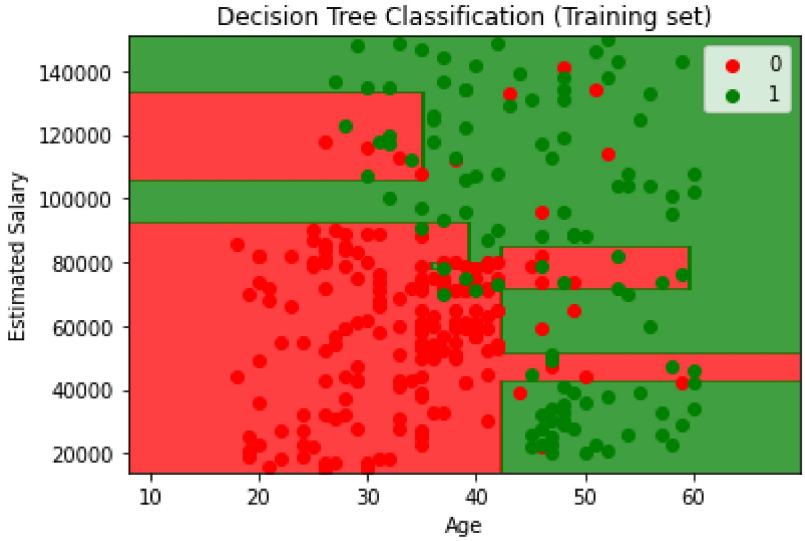


In [15]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(30, 10))
plot_tree(
    classifier,
    feature_names=['Age',"EstimatedSalary"],
    impurity=False,
    filled=True,
    fontsize=17
)
plt.show()
```



Visualizing the Training set results

In [16]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set
[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.
ravel()]).T)).reshape(X1.shape),
             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()
```



Visualizing the Test set results

In [17]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:,
0].max() + 10, step = 0.25),
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set
[:, 1].max() + 1000, step = 0.25))
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.
ravel()]).T)).reshape(X1.shape),
             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()
```