Experiment:

AIM: Write a program to implement decision tree using ID3 and CART algorithm.

THEORY :

Decision Tree : Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Construction of Decision Tree :
A tree can be *"learned"* by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called *recursive partitioning*. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

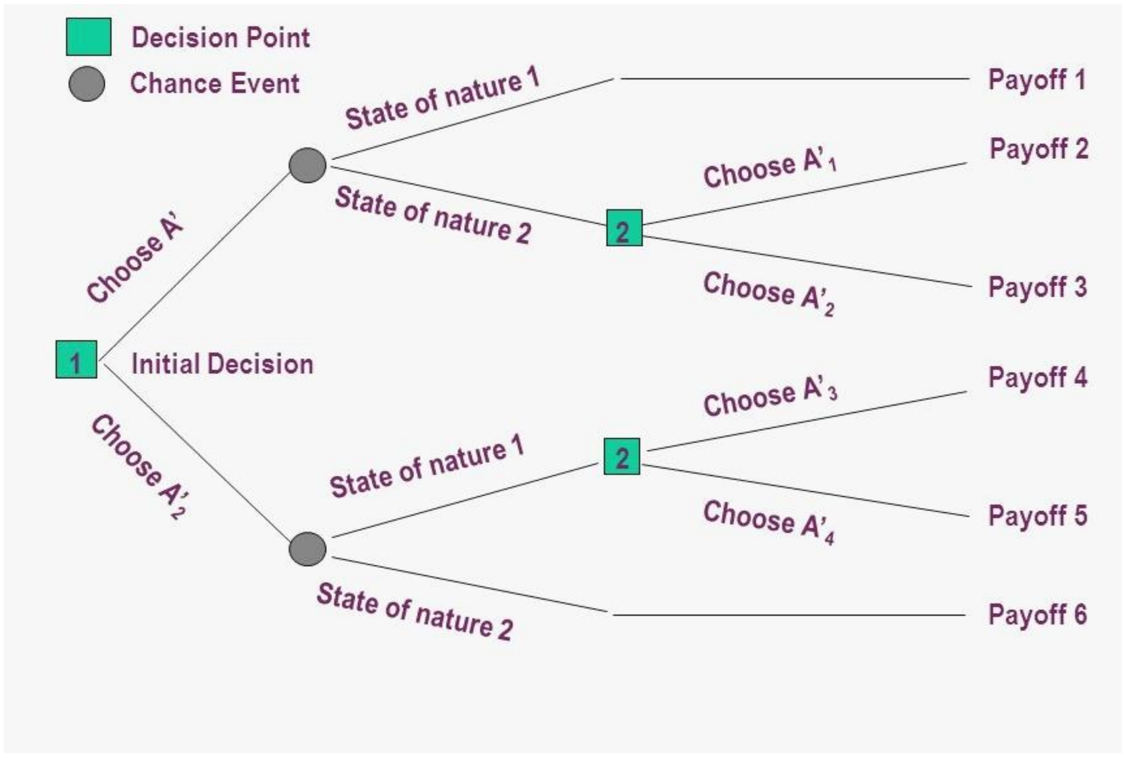
Decision Tree Representation :
Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown. This process is then repeated for the subtree rooted at the new node.

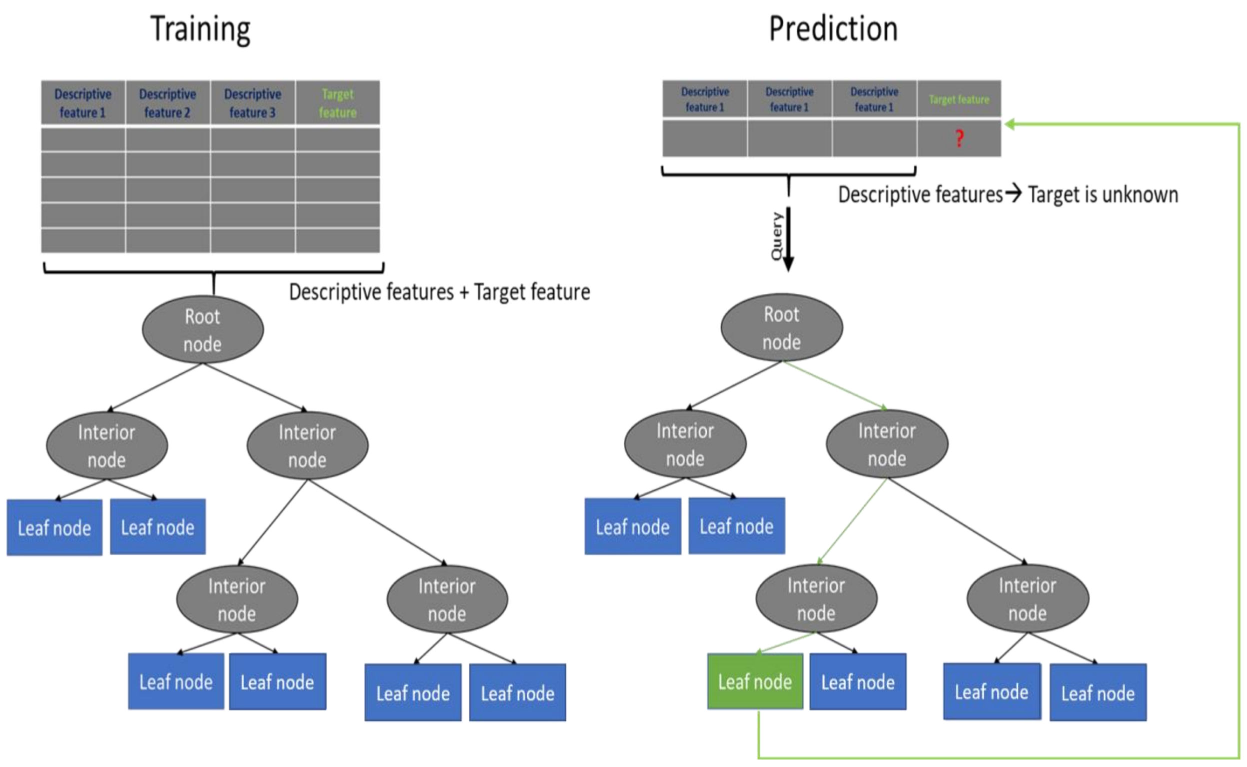
Decision trees are supervised learning algorithms used for both, classification and regression tasks where we will concentrate on classification in this first part of our decision tree tutorial. Decision trees are assigned to the information based learning algorithms which use different measures of information gain for learning. We can use decision trees for issues where we have continuous but also categorical input and target features. The main idea of decision trees is to find those descriptive features which contain the most "information" regarding the target feature and then split the dataset along the values of these features such that the target feature values for the resulting sub_datasets are as pure as possible --> The descriptive feature which leaves the target feature most purely is said to be the most informative one. This process of finding the "most informative" feature is done until we accomplish a stopping criteria where we then finally end up in so called **leaf nodes**.

The leaf nodes contain the predictions we will make for new query instances presented to our trained model. This is possible since the model has kind of learned the underlying structure of the training data and hence can, given some assumptions, make predictions about the target feature value (class) of unseen query instances.

A decision tree mainly contains of a **root node**, **interior nodes**, and **leaf nodes** which are then connected by **branches**.

GENERAL FORMAT OF A DECISION TREE





We split the dataset on the basis of some value possessed by a given attribute, In ID3 algorithm we use Entropy and Information gain and we use ginni index to split the dataset in CART algorithm.

3.1 ID3

THEORY :

The ID3 algorithm begins with the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy E(S) (or information gain IG(A)) of that attribute. It then selects the attribute whichhas the smallest entropy (or largest information gain) value. The set S is then split or partitioned by the selected attribute to produce subsets of the data. (For example, a node can be split into child nodes based upon the subsets of the population whose ages are less than 50, between 50 and 100, and greater than 100.) The algorithm continues to recur on each subset, considering only attributes never selected before.

Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class; in which case the node is turned into a leaf node and labelled with the class of the examples.
- there are no more attributes to be selected, but the examples still do not belong to the same class. In this case, the node is made a leaf node and labelled with the most common class of the examples in the subset.
- there are no examples in the subset, which happens when no example in the parent set was found to match a specific value of the selected attribute. An example could be the absence of a person among the population with age over 100 years. Then a leaf node is created and labelled with the most common class of the examples in the parent node's set.

Throughout the algorithm, the decision tree is constructed with each non-terminal node (internal node) representing the selected attribute on which the data was split, and terminal nodes (leaf nodes) representing the class label of the final subset of this branch.

Entropy:

Entropy is a formula to calculate the homogeneity of a sample. A completely homogenous sample has entropy of 0. An equally divided sample has entropy 1. The formula for entropy is:

$\text{Entropy}(S) = -p(i) * \log_2(p(i))$,where, $p(i)$ is the probability of S belonging to class i.

Information Gain :

The information gain is based on the decrease in entropy after dataset is split on an attribute. The formula for calculating information gain is:

$\text{Gain}(S,A) = \text{Entropy}(S) - ((|S_v|/|S|) * \text{Entropy}(S_v))$

where,S_v = subset of S for which attribute A has value v. $|S_v|$ = number of elements in S_v

$|S|$ = number of elements in S

ALGORITHM :

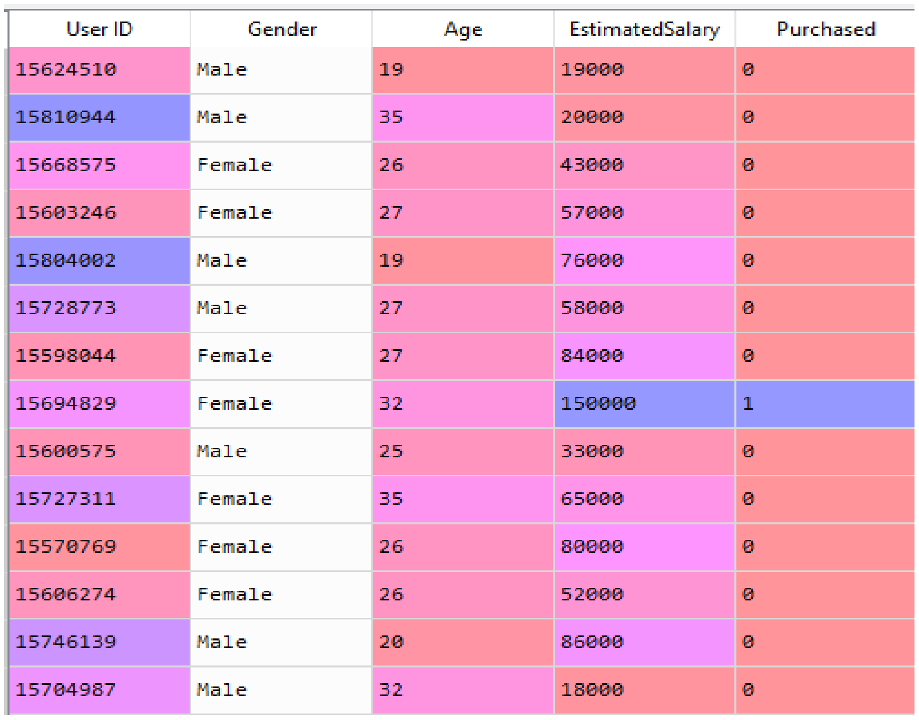
```
ID3(D,Feature_Attributes,Target_Attributes)
    Create a root node r
    Set r to the mode target feature value in D
    If all target feature values are the same:
        return r
    Else:
        pass
    If Feature_Attributes is empty:
        return r
    Else:
        Att = Attribute from Feature_Attributes with the largest information gain value
        r = Att
        For values in Att:
            Add a new node below r where node_values = (Att == values)
            Sub_D_values = (Att == values)
            If Sub_D_values == empty:
                Add a leaf node l where l equals the mode target value in D
            Else:
                add Sub_Tree with ID3(Sub_D_values,Feature_Attributes = Feature_Attributes without Att,
Target_Attributes)
```

CODE:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv(‘sales.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation 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
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

#Printing the classification report
from sklearn.metrics import classification_report,accuracy_score
print(classification_report(y_test, y_pred))
print( "Accuracy is ", accuracy_score(y_test,y_pred)*100)

# 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, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))
```

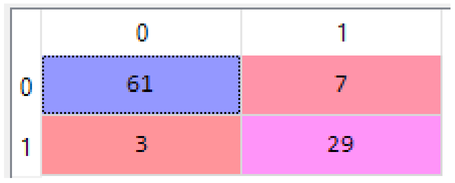
```
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('yellow', 'brown'))(i), label = j)
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
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, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))
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(('yellow', 'brown'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

import pydotplus

from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
dot_data = StringIO()
export_graphviz(classifier,out_file=dot_data,filled=True,rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

OUTPUT :



User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0

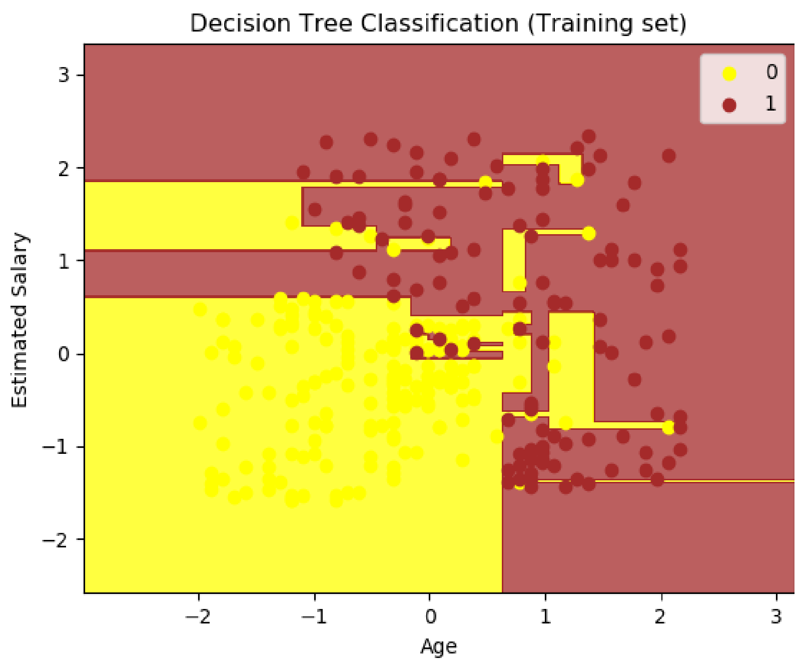
CONFUSION MATRIX

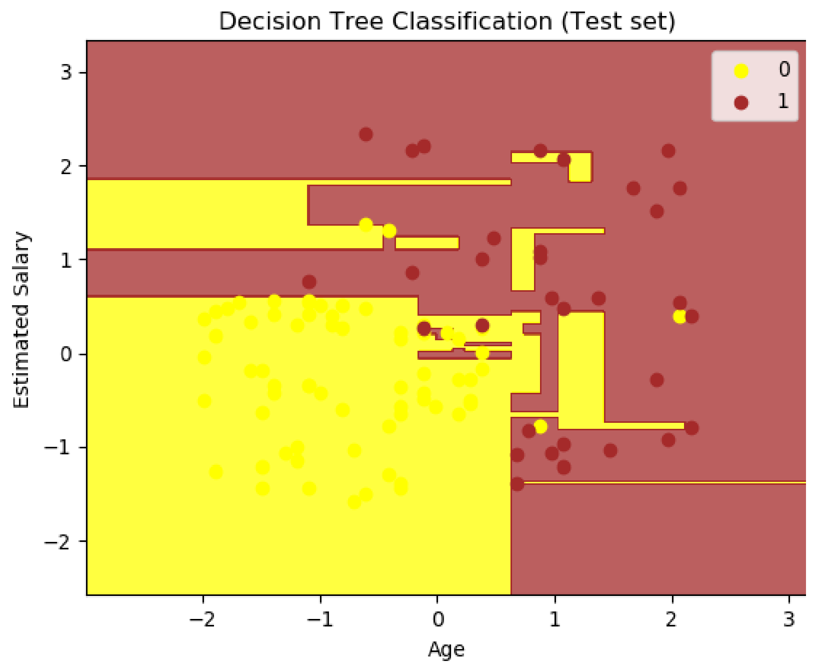


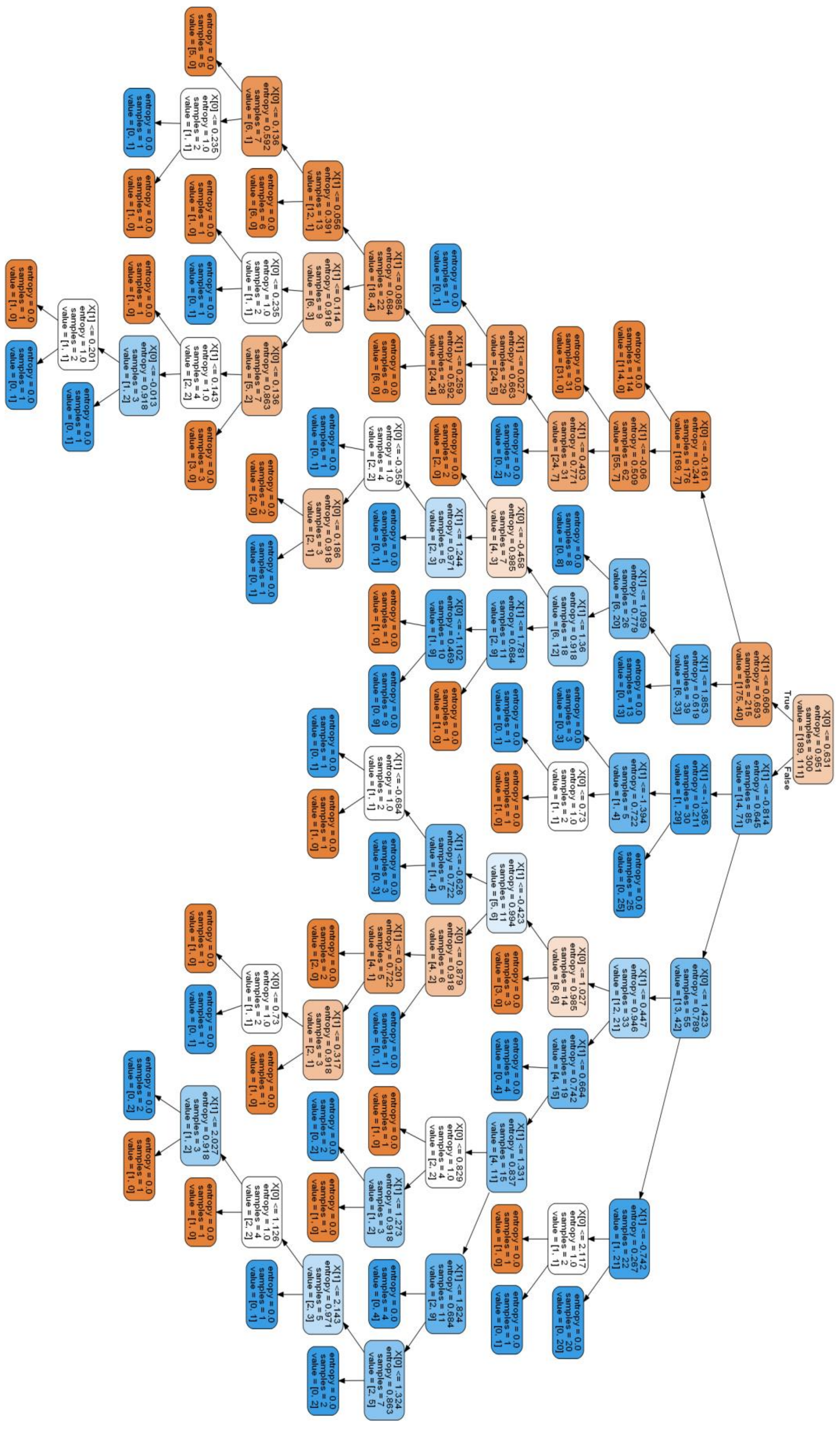
	0	1
0	61	7
1	3	29

```
             precision    recall  f1-score   support

          0       0.95      0.90      0.92        68
          1       0.81      0.91      0.85        32

avg / total       0.91      0.90      0.90       100
```

Accuracy is 90.0


Decision Tree Classification (Training set)
Estimated Salary
Age
0
1


Decision Tree Classification (Test set)
Estimated Salary
Age
0
1


X[0] <= 0.631
entropy = 0.951
samples = 300
value = [189, 111]
True
False

3.2 CART

THEORY :

Creating a CART model involves selecting input variables and split points on those variables until a suitable tree is constructed.
The selection of which input variable to use and the specific split or cut-point is chosen using a greedy algorithm to minimize a cost function. Tree construction ends using a predefined stopping criterion, such as a minimum number of training instances assigned to each leaf node of the tree.

Greedy Splitting
Creating a binary decision tree is actually a process of dividing up the input space. A greedy approach is used to divide the space called recursive binary splitting.
This is a numerical procedure where all the values are lined up and different split points are tried and tested using a cost function. The split with the best cost (lowest cost because we minimize cost) is selected.

All input variables and all possible split points are evaluated and chosen in a greedy manner (e.g. the very best split point is chosen each time).
For regression predictive modeling problems the cost function that is minimized to choose split points is the sum squared error across all training samples that fall within the rectangle:

sum(y – prediction)^2

Where y is the output for the training sample and prediction is the predicted output for the rectangle.

For classification the Gini index function is used which provides an indication of how “pure” the leaf nodes are (how mixed the training data assigned to each node is).

G = sum(pk * (1 – pk))

Where G is the Gini index over all classes, pk are the proportion of training instances with class k in the rectangle of interest.
A node that has all classes of the same type (perfect class purity) will have G=0, where as a G that has a 50-50 split of classes for a binary classification problem (worst purity) will have a G=0.5.
For a binary classification problem, this can be re-written as:
G = 2 * p1 * p2
or
G = 1 – (p1^2 + p2^2)

The Gini index calculation for each node is weighted by the total number of instances in the parent node. The Gini score for a chosen split point in a binary classification problem is therefore calculated as follows:

G = ((1 – (g1_1^2 + g1_2^2)) * (ng1/n)) + ((1 – (g2_1^2 + g2_2^2)) * (ng2/n))

Where G is the Gini index for the split point, g1_1 is the proportion of instances in group 1 for class 1, g1_2 for class 2, g2_1 for group 2 and class 1, g2_2 group 2 class 2, ng1 and ng2 are the total number of instances in group 1 and 2 and n are the total number of instances we are trying to group from the parent node.

Stopping Criterion

The recursive binary splitting procedure described above needs to know when to stop splitting as it works its way down the tree with the training data.
The most common stopping procedure is to use a minimum count on the number of training instances assigned to each leaf node. If the count is less than some minimum then the split is not accepted and the node is taken as a final leaf node.
The count of training members is tuned to the dataset, e.g. 5 or 10. It defines how specific to the training data the tree will be. Too specific (e.g. a count of 1) and the tree will overfit the training data and likely have poor performance on the test set.

Pruning The Tree

The stopping criterion is important as it strongly influences the performance of your tree. You can use pruning after learning your tree to further lift performance.
The complexity of a decision tree is defined as the number of splits in the tree. Simpler trees are preferred. They are easy to understand (you can print them out and show them to subject matter experts), and they are less likely to overfit your data.
The fastest and simplest pruning method is to work through each leaf node in the tree and evaluate the effect of removing it using a hold-out test set. Leaf nodes are removed only if it results in a drop in the overall cost function on the entire test set. You stop removing nodes when no further improvements can be made.
More sophisticated pruning methods can be used such as cost complexity pruning (also called weakest link pruning) where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree.

ALGORITHM :

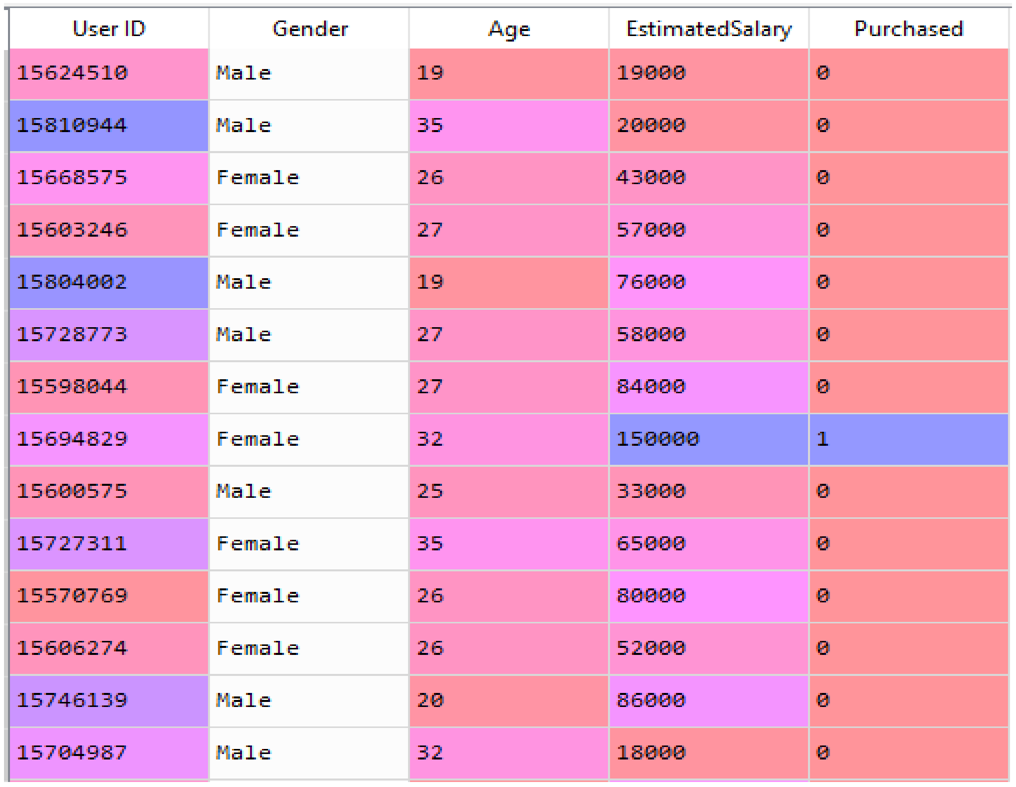
```
CART (D,Feature_Attributes,Target_Attributes)
    Create a root node r
    Set r to the mode target feature value in D
    If all target feature values are the same:
        return r
    Else:
        pass
    If Feature_Attributes is empty:
        return r
    Else:
        Att = Attribute from Feature_Attributes with the smallest weighted ginni index value
        r = Att
        For values in Att:
            Add a new node below r where node_values = (Att == values)
            Sub_D_values = (Att == values)
            If Sub_D_values == empty:
                Add a leaf node l where l equals the mode target value in D
            Else:
                add Sub_Tree with ID3(Sub_D_values,Feature_Attributes = Feature_Attributes without Att,
Target_Attributes)
```

CODE:

```
# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


# Importing the dataset

dataset = pd.read_csv('sales.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values


# Splitting the dataset into the Training set and Test set

from sklearn.cross_validation 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

sc = StandardScaler()

X_train = sc.fit_transform(X_train)

X_test = sc.transform(X_test)


# Fitting Decision Tree Classification to the Training set

from sklearn import tree

classifier = tree.DecisionTreeClassifier(criterion = "gini", random_state = 100,

                                max_depth=3, min_samples_leaf=5)
```

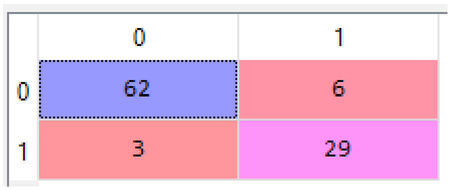
```
classifier = classifier.fit(X_train, y_train)


# Predicting the Test set results

y_pred = classifier.predict(X_test)


# Making the Confusion Matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)


#Printing the classification report

from sklearn.metrics import classification_report,accuracy_score

print(classification_report(y_test, y_pred))

print( "Accuracy is ", accuracy_score(y_test,y_pred)*100)


# 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, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))

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(('yellow', 'brown'))(i), label = j)
plt.title('Decision Tree Classification (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, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('yellow', 'brown')))
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(('yellow', 'brown'))(i), label = j)
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()



import pydotplus

from sklearn.externals.six import StringIO

from IPython.display import Image

from sklearn.tree import export_graphviz

dot_data = StringIO()

export_graphviz(classifier,out_file=dot_data,filled=True,rounded=True)

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())

Image(graph.create_png())
```

OUTPUT :



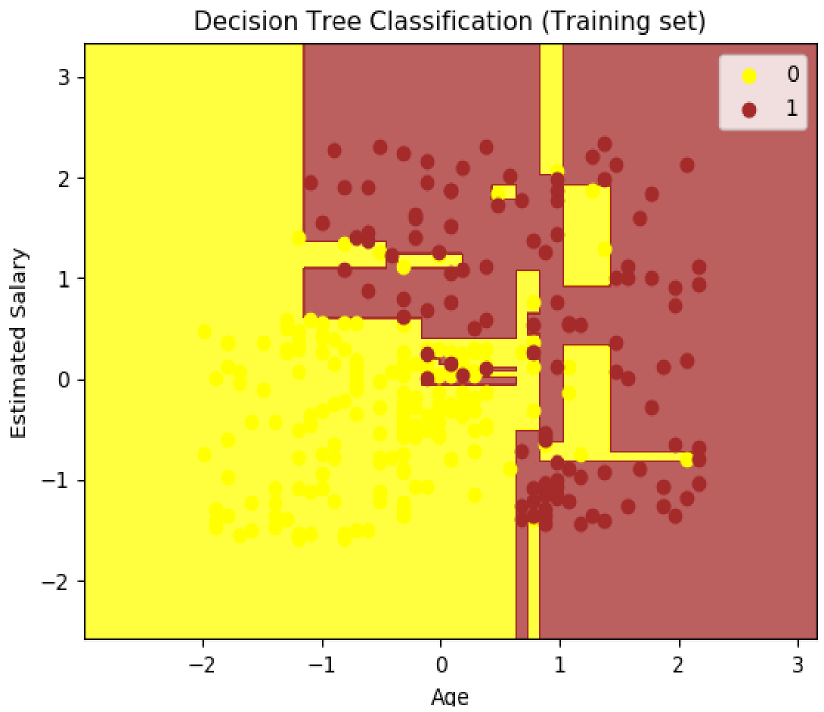
User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0

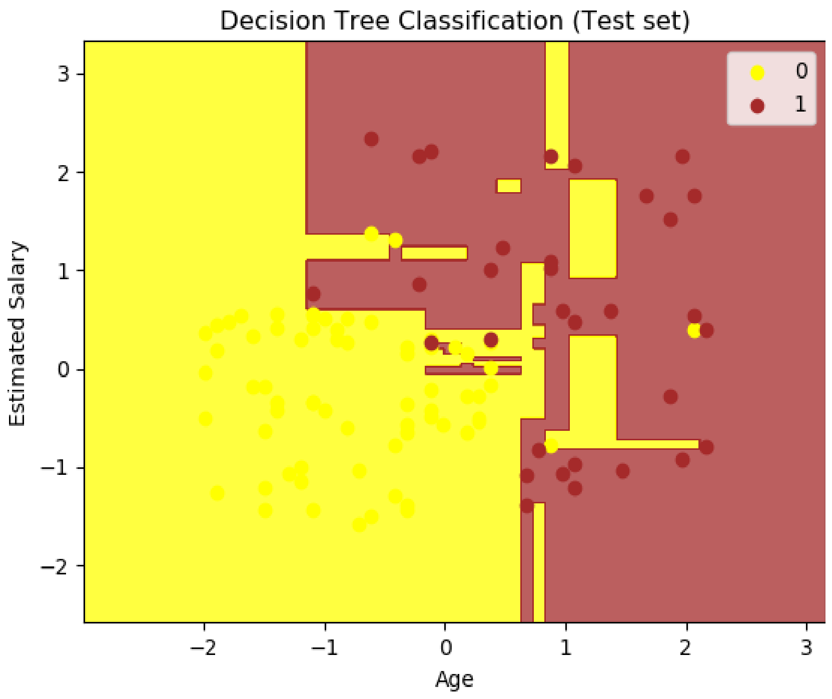
Confusion matrix

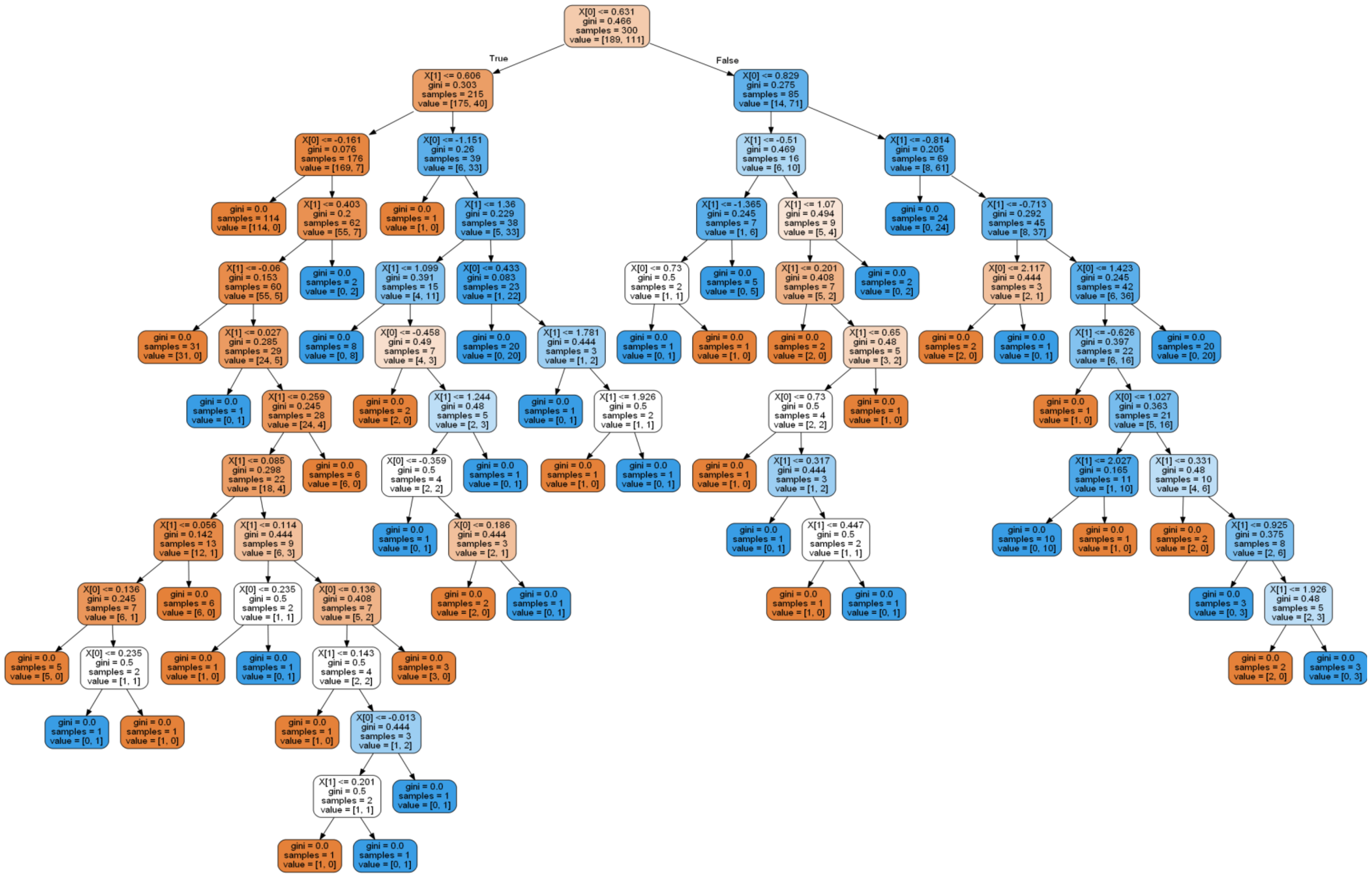


	0	1
0	62	6
1	3	29

```
             precision    recall  f1-score   support

          0       0.95      0.91      0.93        68
          1       0.83      0.91      0.87        32

avg / total       0.91      0.91      0.91       100
Accuracy is  91.0
```


Decision Tree Classification (Training set)
Estimated Salary
Age
0
1
−2
−1
0
1
2
3


Decision Tree Classification (Test set)
Estimated Salary
Age
0
1
−2
−1
0
1
2
3


X[0] <= 0.631 gini = 0.466 samples = 300 value = [189, 111]
True
False
X[1] <= 0.606 gini = 0.303 samples = 215 value = [175, 40]
X[0] <= 0.829 gini = 0.275 samples = 85 value = [14, 71]
X[0] <= -0.161 gini = 0.076 samples = 176 value = [169, 7]
X[0] <= -1.151 gini = 0.26 samples = 39 value = [6, 33]
X[1] <= -0.51 gini = 0.469 samples = 16 value = [6, 10]
X[1] <= -0.814 gini = 0.205 samples = 69 value = [8, 61]
gini = 0.0 samples = 114 value = [114, 0]
X[1] <= 0.403 gini = 0.2 samples = 62 value = [55, 7]
gini = 0.0 samples = 1 value = [1, 0]
X[1] <= 1.36 gini = 0.229 samples = 38 value = [5, 33]
X[1] <= -1.365 gini = 0.245 samples = 7 value = [1, 6]
X[1] <= 1.07 gini = 0.494 samples = 9 value = [5, 4]
gini = 0.0 samples = 24 value = [0, 24]
X[1] <= -0.713 gini = 0.292 samples = 45 value = [8, 37]
X[1] <= -0.06 gini = 0.153 samples = 60 value = [55, 5]
gini = 0.0 samples = 2 value = [0, 2]
X[1] <= 1.099 gini = 0.391 samples = 15 value = [4, 11]
X[0] <= 0.433 gini = 0.083 samples = 23 value = [1, 22]
X[0] <= 0.73 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 5 value = [0, 5]
X[1] <= 0.201 gini = 0.408 samples = 7 value = [5, 2]
gini = 0.0 samples = 2 value = [0, 2]
X[0] <= 2.117 gini = 0.444 samples = 3 value = [2, 1]
X[0] <= 1.423 gini = 0.245 samples = 42 value = [6, 36]
gini = 0.0 samples = 31 value = [31, 0]
X[1] <= 0.027 gini = 0.285 samples = 29 value = [24, 5]
gini = 0.0 samples = 8 value = [0, 8]
X[0] <= -0.458 gini = 0.49 samples = 7 value = [4, 3]
gini = 0.0 samples = 20 value = [0, 20]
X[1] <= 1.781 gini = 0.444 samples = 3 value = [1, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 2 value = [2, 0]
X[1] <= 0.65 gini = 0.48 samples = 5 value = [3, 2]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 1 value = [0, 1]
X[1] <= -0.626 gini = 0.397 samples = 22 value = [6, 16]
gini = 0.0 samples = 20 value = [0, 20]
gini = 0.0 samples = 1 value = [0, 1]
X[1] <= 0.259 gini = 0.245 samples = 28 value = [24, 4]
gini = 0.0 samples = 2 value = [2, 0]
X[1] <= 1.244 gini = 0.48 samples = 5 value = [2, 3]
gini = 0.0 samples = 1 value = [0, 1]
X[1] <= 1.926 gini = 0.5 samples = 2 value = [1, 1]
X[0] <= 0.73 gini = 0.5 samples = 4 value = [2, 2]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [1, 0]
X[0] <= 1.027 gini = 0.363 samples = 21 value = [5, 16]
X[1] <= 0.085 gini = 0.298 samples = 22 value = [18, 4]
gini = 0.0 samples = 6 value = [6, 0]
X[0] <= -0.359 gini = 0.5 samples = 4 value = [2, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
X[1] <= 0.317 gini = 0.444 samples = 3 value = [1, 2]
X[1] <= 2.027 gini = 0.165 samples = 11 value = [1, 10]
X[1] <= 0.331 gini = 0.48 samples = 10 value = [4, 6]
X[1] <= 0.056 gini = 0.142 samples = 13 value = [12, 1]
X[1] <= 0.114 gini = 0.444 samples = 9 value = [6, 3]
gini = 0.0 samples = 1 value = [0, 1]
X[0] <= 0.186 gini = 0.444 samples = 3 value = [2, 1]
gini = 0.0 samples = 1 value = [0, 1]
X[1] <= 0.447 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 10 value = [0, 10]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 2 value = [2, 0]
X[1] <= 0.925 gini = 0.375 samples = 8 value = [2, 6]
X[0] <= 0.136 gini = 0.245 samples = 7 value = [6, 1]
gini = 0.0 samples = 6 value = [6, 0]
X[0] <= 0.235 gini = 0.5 samples = 2 value = [1, 1]
X[0] <= 0.136 gini = 0.408 samples = 7 value = [5, 2]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 3 value = [0, 3]
X[1] <= 1.926 gini = 0.48 samples = 5 value = [2, 3]
gini = 0.0 samples = 5 value = [5, 0]
X[0] <= 0.235 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]
X[1] <= 0.143 gini = 0.5 samples = 4 value = [2, 2]
gini = 0.0 samples = 3 value = [3, 0]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [1, 0]
X[0] <= -0.013 gini = 0.444 samples = 3 value = [1, 2]
X[1] <= 0.201 gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 1 value = [0, 1]

DISCUSSION :

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its widely used in machine learning.

ADVANTAGES :

- White box, easy to interpret model
- No feature normalization needed
- Tree models can handle both continuous and categorical data (Classification and Regression Trees)
- Can model nonlinear relationships
- Can model interactions between the different descriptive features

DISADVANTAGES:

- If continuous features are used the tree may become quite large and hence less interpretable
- Decision trees are prone to overfit the training data and hence do not well generalize the data if no stopping criteria or improvements like pruning, boosting or bagging are implemented
- Small changes in the data may lead to a completely different tree. This issue can be addressed by using ensemble methods like bagging, boosting or random forests

FINDING AND LEARNING :

In this experiment we implemented Decision Tree algorithm using sklearn library in python on a sales dataset which classifies whether a person will by a certain product given his/her age and salary. And We got an accuracy of 90%(With ID3 algorithm) and 91%.(With CART algorithm).