INF-552
MACHINE LEARNING FOR DATA SCIENCE

PROGRAMMING ASSIGNMENT 1:
DECISION TREES

NAME	USC-ID	EMAIL
Amitabh Rajkumar Saini	7972003272	amitabhr@usc.edu
Shilpa Jain	4569044628	shilpaj@usc.edu
Sushumna Khandelwal	7458911214	sushumna@usc.edu

PART 1: IMPLEMENTATION

- **Data Structures Defined/Used**
  - Dataset represented as numpy 2-D Array
  - Decision Node
    A decision node defines the decision factor/attribute, the information gain and other meta-data.
    Following is full structure of a Decision Node.

Decision_Node
+ df: numpy 2-D Array + attributes: list() + df_entropy: float + infogain: float + parent_node: Decision_Node + children_nodes: dict() + node_name: String
+ entropy(): float + calcinfogain(): float + get_best_split(): list(numpy 2-D Array) + create_child(): Decision_Node

  - Decision Tree
    A decision tree contains methods to create the tree using ID3 Algorithm, predict and export the tree as directed graph.
    Following is the full structure of a Decision Tree.

Decision_Tree
+ root: Decision_Node + full_df: numpy 2-D Array
+ csv_to_df(): numpy 2-D Array + create_tree(): void + print_tree(): void + predict(): list(str) + create_graph(): void

- **Code: (Language - Python)**

```
'''

PA-1: Decision Trees
Authors:
Amitabh Rajkumar Saini, amitabhr@usc.edu
Shilpa Jain, shilpaj@usc.edu
Sushumna Khandelwal, sushumna@usc.edu

Dependencies:
1. numpy: pip install numpy
2. graphviz: pip install graphviz
3. graphviz for OS: https://graphviz.gitlab.io/download/

Output:
Returns a decision tree graph as PNG File and the prediction on console.

'''

import numpy as np
import re
from graphviz import Digraph
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'

class decision_node:
    '''
    Decision Node defines a single node of the tree containing all meta-data
    '''
    def __init__(self,df,parent):
        '''
        Constructs a decision_node
        :param df: Dataframe of type numpy 2D Array
        :param parent: Parent of this node of type decision_node
        :return: returns nothing
        '''
        self.df = df[1:]
        self.attributes = df[0]
        self.df_entropy = float('-inf')
        self.infogain = float('-inf')
        self.parent_node = parent
        self.children_nodes = dict()
        self.node_name = ""


    def entropy(self,label_index,attribute_index=None):
        '''
        Calculates entropy of a particular attribute or the whole dataframe
        :param label_index: index of label column
        :param attribute_index: index of attribute column
        :return: returns entropy
        '''
        if attribute_index == None:
            values,count = np.unique(self.df[:,label_index],return_counts =
    True)
            e = 0
            for each in count:
                p = each/np.sum(count)
                e -= p * np.log2(p)
            return e
```

```
    else:
        values,count = np.unique(self.df[:,attribute_index],
return_counts=True)
        e_attr = 0
        attr_value_indices={}
        for i in range(count.size):
            p_attr = count[i]/np.sum(count)
            indices = np.where(self.df[:,attribute_index]==values[i])
            attr_value_indices[values[i]]=indices
            attr_label,attr_label_count =
np.unique(np.take(self.df[:,label_index],indices),return_counts=True)
            e = 0
            for each in attr_label_count:
                p = each/np.sum(attr_label_count)
                e -= p * np.log2(p)
            e_attr += p_attr*e
        return e_attr, attr_value_indices

def calcinfogain(self,cur_df_entropy,attr_entropy):
    '''
    Calculates information gain
    :param cur_df_entropy: Entropy of DataFrame
    :param attr_entropy: Entropy of the Attribute
    :return: returns information gain
    '''
    return cur_df_entropy-attr_entropy

def get_best_split(self):
    '''
    Finds the best attribute to split the data using information gain
    :return: returns dataframe after splitting on the attribute and the
attribute
    '''
    max_gain = float('-inf')
    max_attribute_index = -1
    max_attr_df_indices = np.asarray([])
    for i in range(self.df.shape[1]-1):           #no.of cols
        attr_entropy, attr_df_indices = self.entropy(-1,i)
        self.df_entropy = self.entropy(-1)
        gain = self.calcinfogain(self.df_entropy,attr_entropy)
        if max_gain < gain:
            max_gain = gain
            max_attribute_index = i
            max_attr_df_indices = attr_df_indices
    self.infogain = max_gain
    if self.infogain == 0 or self.infogain == float('-inf'):
        values,count = np.unique(self.df[:,-1], return_counts=True)
        maxcount = float('-inf')
        for i in range(count.size):
            if maxcount<count[i]:
                maxcount=count[i]
                self.node_name = values[i]

        return None,[]
    self.node_name = self.attributes[max_attribute_index]
    return max_attribute_index, max_attr_df_indices

def create_child(self,col_index,attr_name,attr_df_indices):
    '''
    Creates a child node
    :param col_index: index of the attribute column used for splitting
    :param attr_name: attribute value for which the node will be created
```

```
        :param attr_df_indices: indices of the rows containig the attr_name
        :return: returns child node
        '''
        t_df = self.df[attr_df_indices]
        t_df = np.delete(t_df,col_index,axis=1)
        h_df = np.delete(self.attributes,col_index)
        x = []
        x.append(h_df)
        df = np.append(np.asarray(x),t_df,axis=0)
        child_node = decision_node(df,self)
        self.children_nodes[attr_name] = child_node
        return child_node

class decision_tree:
    '''
    Class contains methods to create the tree using ID3 Algorithm, predict and
     export the tree as directed graph
    '''
    def __init__(self):
        '''
        Creates a decision tree instance
        :return: returns nothing
        '''
        self.full_df = None
        self.root = None
        #self.attributes = None

    def txt_to_df(self,filepath):
        '''
        Opens txt file and makes numpy 2d array
        :return: returns numpy 2d array
        '''
        f=open(filepath,"r")
        lines = f.readlines()

        def clean_data(each_line):
            each_line = each_line.split(":")[-1]
            each_line = re.sub('[^0-9a-zA-Z,\- ]+', '', each_line)
            return each_line
        cleaned_data = []
        for i, each in enumerate(lines):
            if not re.match(r'^\s*$', each):
                cleaned_data.append(clean_data(each))
        df = []
        for i in range(len(cleaned_data)):
            df.append(cleaned_data[i].strip().replace(", ",",").split(","))
        return np.asarray(df)

    def csv_to_df(self,filepath):
        '''
        Opens csv file and makes numpy 2d array
        :return: returns numpy 2d array
        '''
        f=open(filepath,"r")
        lines = f.readlines()

        df = []
        for i in range(len(lines)):
            df.append(lines[i].strip().split(","))
        return np.asarray(df)

    def create_tree(self):
```

```
        '''
        Creates decision tree using ID3 algorithm iteratively
        :return: returns nothing
        '''
        self.root = decision_node(self.full_df,None)
        q = []
        q.append(self.root)
        while(q):
            d_node = q.pop(0)
            attr_index, max_attr_df_indices = d_node.get_best_split()
            for each in max_attr_df_indices:

    q.append(d_node.create_child(attr_index,each,max_attr_df_indices[each]))


    def print_tree(self):
        '''
        Prints decision tree on console
        :return: returns nothing
        '''
        q = []
        q.append((self.root,None))
        while(q):
            x = q.pop(0)
            p = x[0].parent_node.node_name if x[0].parent_node!=None else
    'None'
            print(p,x[0].node_name,x[0].infogain,x[1])
            for each in x[0].children_nodes:
                q.append((x[0].children_nodes[each],each))

    def predict(self,test_df):
        '''
        Predicts using the decision tree generated
        :param test_df: numpy 2d array to be predicted
        :return: returns nothing
        '''
        header = test_df[0]
        out_answers = np.unique(self.root.df[:-1])
        pred_answers = []
        for row in range(1,len(test_df)):
            current = self.root
            row_data = test_df[row]
            while current.node_name not in out_answers:
                index = np.where(header==current.node_name)
                current = current.children_nodes[row_data[index[0][0]]]
            pred_answers.append(current.node_name)
        print(pred_answers)

    def creategraph(self):
        '''
        Creates decision tree using graphviz for visually
        :return: returns nothing
        '''
        graph = Digraph()
        q=[]
        q.append(self.root)

    graph.node(str(self.root),self.root.node_name+"\nIG="+str(round(self.root.i
    nfogain,2))+"\nEntropy="+str(round(self.root.df_entropy,2)))
        while q:
            x = q.pop(0)
            for each in x.children_nodes:
```

```
                graph.node(str(x.children_nodes[each]),x.children_nodes[each].node_name+"\n
IG="+str(round(x.children_nodes[each].infogain,2))+"\nEntropy="+str(round(x
.children_nodes[each].df_entropy,2)))
                graph.edge(str(x),str(x.children_nodes[each]),label=each)
                q.append(x.children_nodes[each])

        graph.render('DecisionTree.gv',view=True,format='png')

def metrics(actual,predicted):
    '''
    Calculates metrics
    :param actual: actual values in list
    :param predicted: model predicted values in list
    :return : returns nothing
    '''
    confusion_matrix=[[0,0],[0,0]]
    for i in range(len(actual)):
        actual[i]=1 if actual[i]=="Yes" else 0
        predicted[i]=1 if predicted[i]=="Yes" else 0

        if actual[i] and predicted[i]:
            confusion_matrix[0][0]+=1
        elif actual[i] and not predicted[i]:
            confusion_matrix[0][1]+=1
        elif not actual[i] and predicted[i]:
            confusion_matrix[1][0]+=1
        else:
            confusion_matrix[1][1]+=1


    accuracy=(confusion_matrix[1][1]+confusion_matrix[0][0])/(sum(confusion_mat
rix[0])+sum(confusion_matrix[1]))
     recall=(confusion_matrix[0][0])/sum(confusion_matrix[0])

    precision=(confusion_matrix[0][0])/(confusion_matrix[0][0]+confusion_matrix
[1][0])
    Fmeasure=(2*recall*precision)/(recall+precision)
    print("Accuracy: ",accuracy)
    print("Recall: ",recall)
    print("Precision: ",precision)
    print("Fmeasure: ",Fmeasure)
    print("=======================")
    Matrix=[]
    Matrix.append(["","P1","N0"])
    Matrix.append((["P1"]+confusion_matrix[0]))
    Matrix.append((["N0"]+confusion_matrix[1]))
    for i in Matrix:
        print(i)
    print("TP: ",confusion_matrix[0][0])
    print("FP: ",confusion_matrix[0][1])
    print("FN: ",confusion_matrix[1][0])
    print("TN: ",confusion_matrix[1][1])

def main():
    '''
    Runner Program
    :return: returns nothing
    '''
    d = decision_tree()
    d.full_df = d.txt_to_df("train.txt")
    d.create_tree()
```

```
    #d.print_tree()
    #print("------------------------")
    pred_df = d.csv_to_df("predict.csv")
    pred = d.predict(pred_df)
    print(pred)
    if pred_df[0,-1] == d.full_df[0,-1]:
        metrics(list(pred_df[1:,-1]),pred)
    d.creategraph()

if __name__== "__main__":
    main()
```

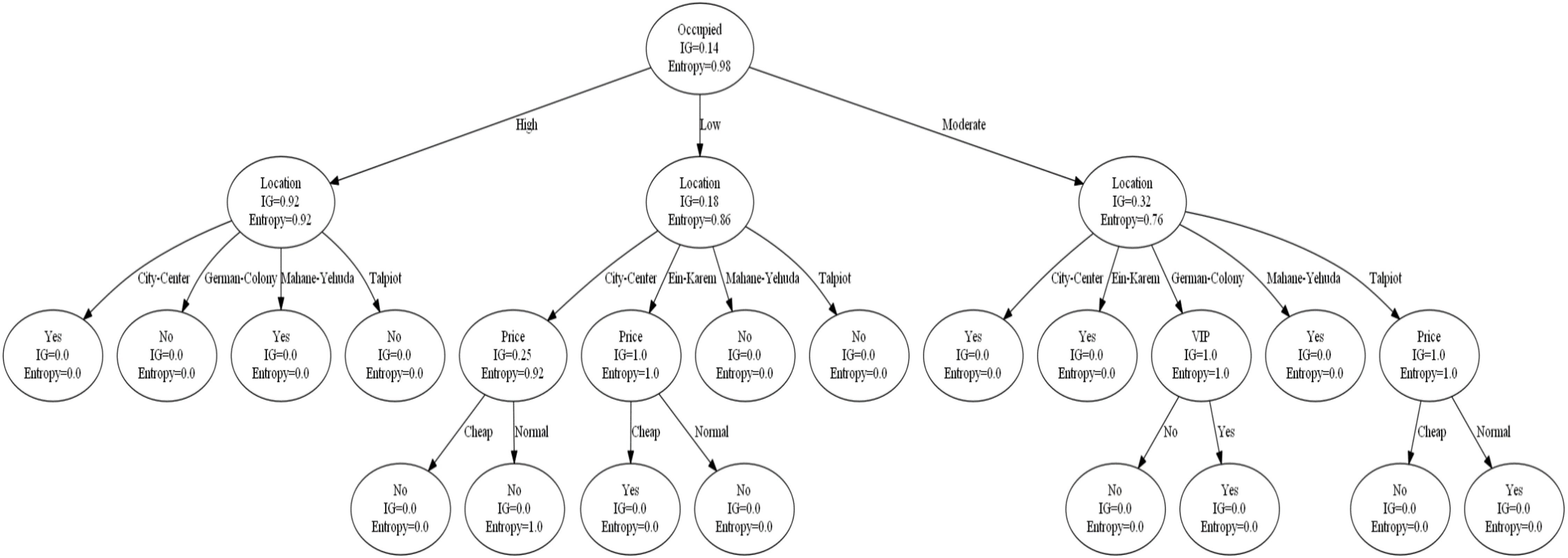
- **Output Format**

  The output tree is generated as PNG file in the current directory using Graphviz

- **Challenges Faced**

1. Deciding the suitable data structures for the model.
2. Faced installation issues with Graphviz.
3. As all of us were coding different parts of algorithm so it initially become difficult for us to integrate each other's code.
4. Evaluating the correctness manually to verify the correctness of Entropy calculated by our algorithm.
5. Rendering the decision tree to visual graph.

- **Run**
  Graph:

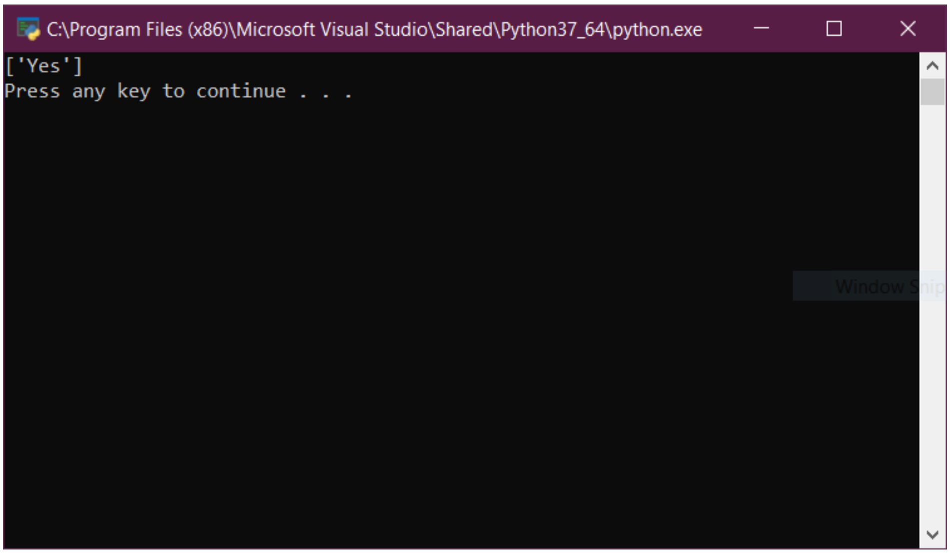


Decision Tree

- **Prediction:**

  occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No

  Predicted Output: Yes



PART 2: SOFTWARE FAMILIARIZATION

Library Function:

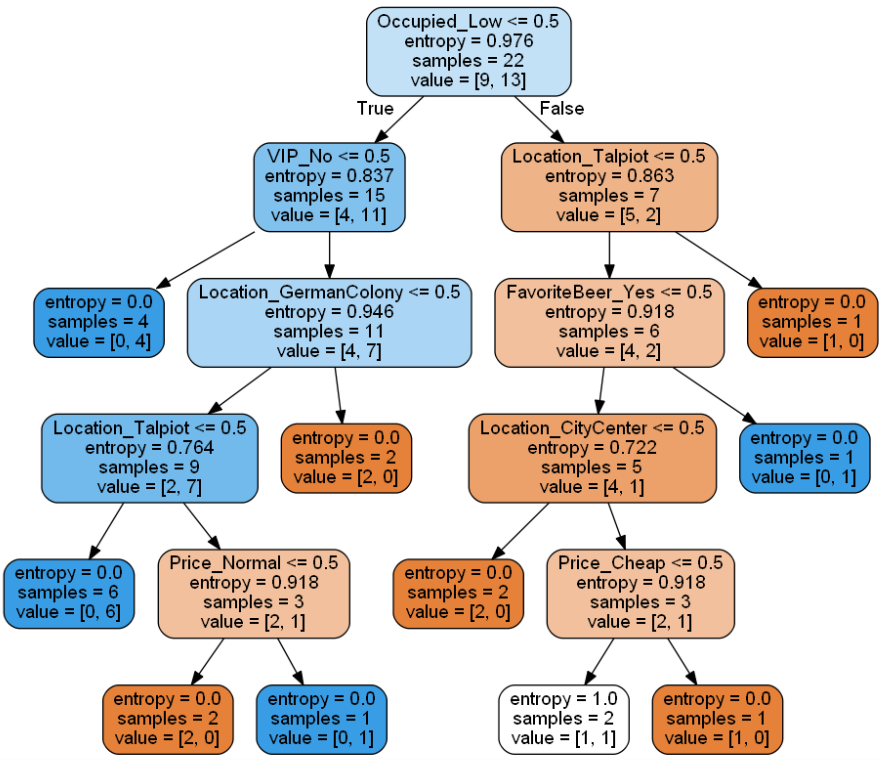
class `sklearn.tree.`**`DecisionTreeClassifier`**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0*)

Implementation:

```
import re
import csv
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
Classifier
from sklearn.model_selection import train_test_split # Import
train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for
accuracy calculation
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus



data = open("data.txt")

complete_data = data.readlines()

def clean_data(each_line):
    each_line = each_line.split(":")[-1]
    each_line = re.sub('[^0-9a-zA-Z,]+', '', each_line)
    return each_line

cleaned_data = []
for i, each in enumerate(complete_data):
    if not re.match(r'^\s*$', each):
        cleaned_data.append(clean_data(each))


with open('cleaned_data.csv', 'w', newline='') as out_file:
    writer = csv.writer(out_file)
    writer.writerows([x.split(",") for x in cleaned_data])
    out_file.close()


df = pd.read_csv("cleaned_data.csv")
feature_col=['Occupied', 'Price', 'Music', 'Location', 'VIP',
'FavoriteBeer']
trim_data=df[feature_col]
trim_data=pd.get_dummies(trim_data)
X = trim_data # Features
y = df.Enjoy # Target variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=1)
clf = DecisionTreeClassifier(criterion='entropy')
clf = clf.fit(X,y)
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
predict_data=pd.DataFrame()
predict_data=predict_data.append({'Occupied_High':0,'Occupied_Low':0,'O
ccupied_Moderate':1,'Price_Cheap':1,'Price_Expensive':0,'Price_Normal':
0,'Music_Loud':1,'Music_Quiet':0,'Location_CityCenter':1,'Location_EinK
arem':0,'Location_GermanColony':0,'Location_MahaneYehuda':0,'Location_T
alpiot':0,'VIP_No':1,'VIP_Yes':0,'FavoriteBeer_No':1,'FavoriteBeer_Yes'
:0}, ignore_index=True)
predicted_data=clf.predict(abc)
print(predicted_data)
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=False,feature_names =
['Occupied_High', 'Occupied_Low', 'Occupied_Moderate', 'Price_Cheap',
       'Price_Expensive', 'Price_Normal', 'Music_Loud', 'Music_Quiet',
       'Location_CityCenter', 'Location_EinKarem',
'Location_GermanColony',
       'Location_MahaneYehuda', 'Location_Talpiot', 'VIP_No',
'VIP_Yes',
       'FavoriteBeer_No', 'FavoriteBeer_Yes'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('image.png')
Image(graph.create_png())
```

Output:



Prediction:

occupied = Moderate; price = Cheap; music = Loud; location = City-Center; VIP = No; favorite beer = No

Output Prediction: Yes

Comparison:

We split the given data in two parts: training and test and gave it to both our and library algorithm.

Training Data and Testing Data is one hot encoded for both the algorithms.

Training Data:

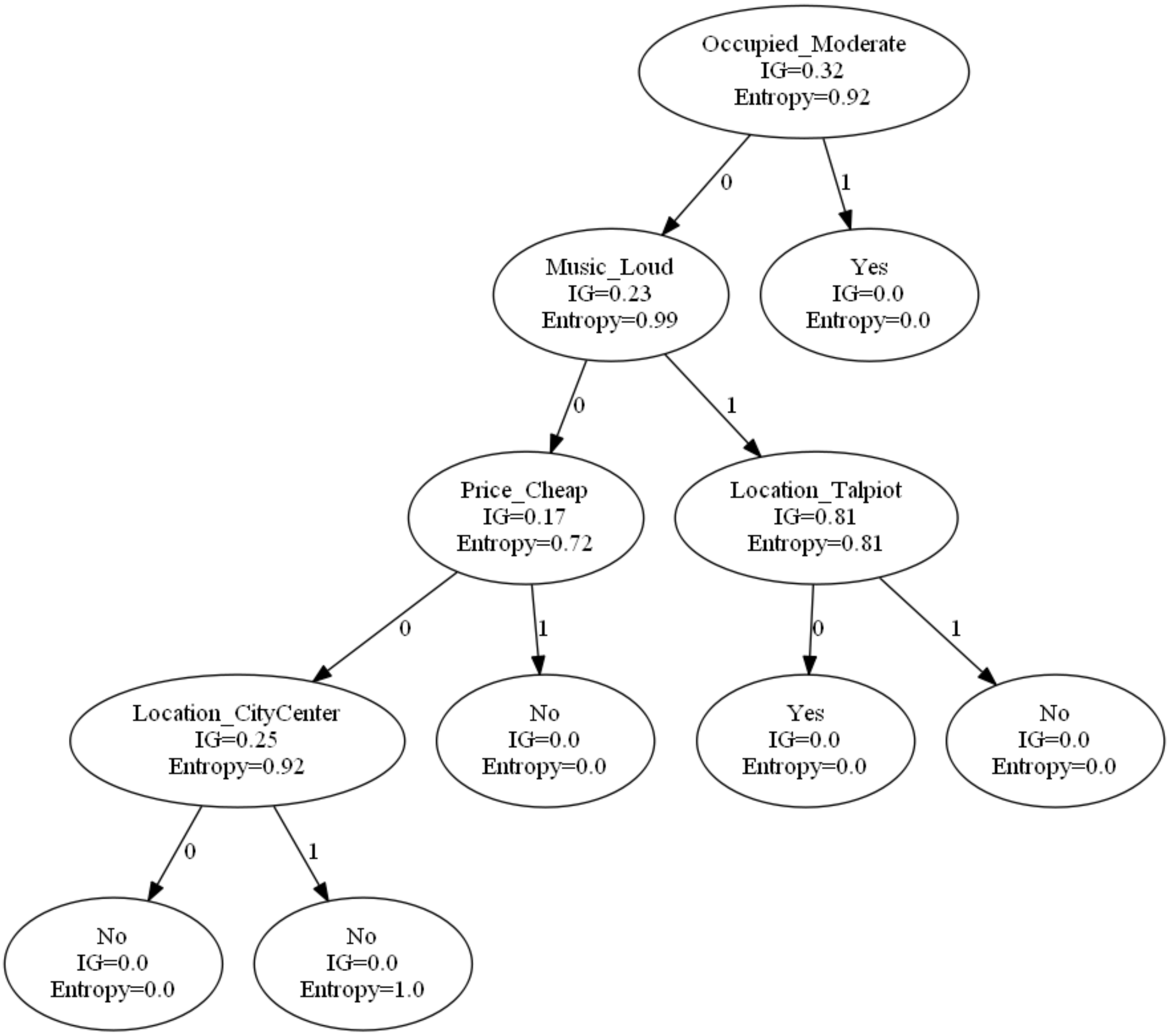
Occupied_High	Occupied_Low	Occupied_Moderate	Price_Cheap	Price_Expensive	Price_Normal	Music_Loud	Music_Quiet	Location_CityCenter	Location_EinKarem	Location_GermanColony	Location_MahaneYehuda	Location_Talpiot	VIP_No	VIP_Yes	FavoriteBeer_No	FavoriteBeer_Yes	Enjoy
0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	Yes
0	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0	1	Yes
0	1	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0	No
0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	No
0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	1	0	Yes
1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	Yes
0	0	1	0	0	1	1	0	0	1	0	0	0	1	0	0	1	Yes
1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	No
0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	0	1	Yes
0	1	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0	Yes
0	1	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	No
1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	0	1	Yes
0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	Yes
0	1	0	1	0	0	0	1	0	0	0	0	1	0	1	0	1	No
0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	Yes

Testing Data:

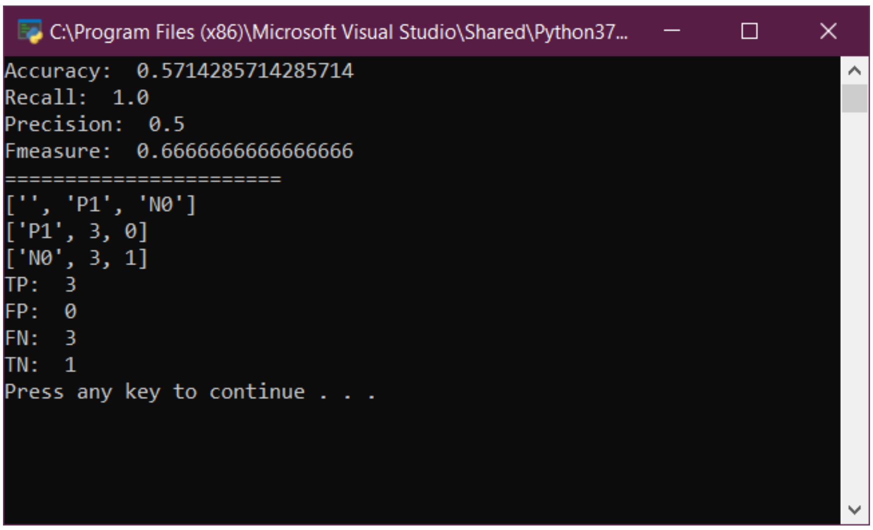
Occupied_High	Occupied_Low	Occupied_Moderate	Price_Cheap	Price_Expensive	Price_Normal	Music_Loud	Music_Quiet	Location_CityCenter	Location_EinKarem	Location_GermanColony	Location_MahaneYehuda	Location_Talpiot	VIP_No	VIP_Yes	FavoriteBeer_No	FavoriteBeer_Yes	Enjoy
0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	0	Yes
1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	Yes
0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	No
1	0	0	0	0	1	1	0	0	0	0	1	0	0	1	0	1	Yes
0	1	0	0	1	0	1	0	0	0	0	1	0	1	0	1	0	No
1	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	No
0	0	1	1	0	0	1	0	0	0	0	0	1	1	0	0	1	No

Following is the output:

1. Our Algorithm
   Tree:



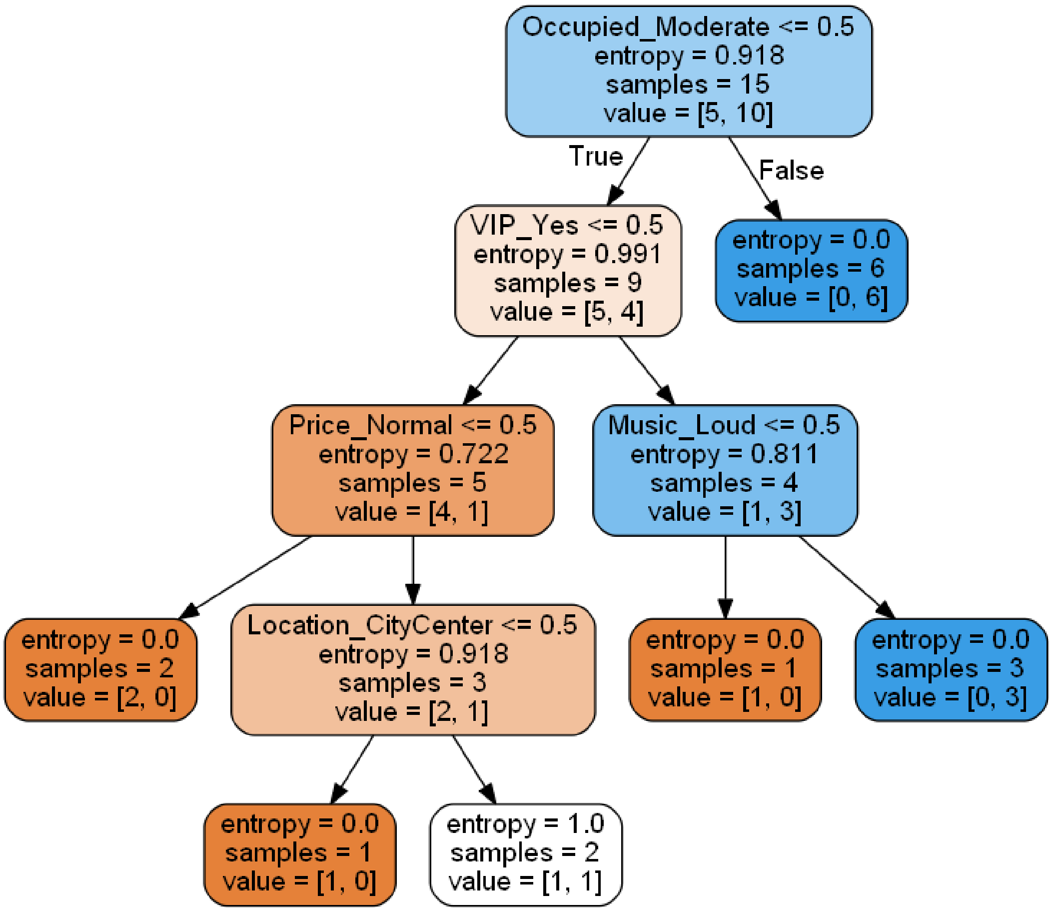
   Metrics:



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37...
Accuracy:  0.5714285714285714
Recall:  1.0
Precision:  0.5
Fmeasure:  0.6666666666666666
=======================
['', 'P1', 'N0']
['P1', 3, 0]
['N0', 3, 1]
TP:  3
FP:  0
FN:  3
TN:  1
Press any key to continue . . .
```

2. Scikit Decision Tree

Tree:



Metrics:

```
Accuracy:  0.5714285714285714
Recall:  0.6666666666666666
Precision:  0.5
Fmeasure:  0.5714285714285715
=======================
['', 'P1', 'NO']
['P1', 2, 1]
['NO', 2, 2]
```

Comments:

1. Our decision tree model only deals with discrete data as compared to the library function which can deal in continuous data as well.
2. We can prune our dataset using library function which could significantly improve the performance, but in our algorithm, we haven't provided the functionality of pruning. This could lead to overfitting of data and poor generalization performance.

PART 3: APPLICATIONS

- **Decision trees are applicable to business operations.** Companies are constantly making decisions regarding issues like product development, staffing, operations, and mergers and acquisitions. Organizing all considered alternatives with a decision tree allows for a simultaneous systematic evaluation of these ideas.
- **Decision Trees in Corporate Analysis**- Decision trees let individuals explore the ranging elements that could materially impact their decisions. Prior to airing a multimillion-dollar Super Bowl commercial, a firm aims to determine the different possible outcomes of their marketing campaign. Various issues can influence the final success or failure of the expenditure, such as the appeal of the commercial, the economic outlook, the quality of the product, and competitors' advertisements. Once the impact of these variables has been determined and the corresponding probabilities assigned, the company can formally decide whether or not to run the ad.
- Prediction of **personality based on social media** using decision tree and categorizing them into Big Five Personality classes: Openness to experience, Conscientiousness, Extraversion, Agreeableness, Neuroticism
- Predict if a **patient has breast cancer or not**, from a blood test; so that the patient could receive very early treatments, even before a tumor becomes noticeable.
- Classification system for **serial criminal patterns (CSSCP)** - using three years' worth of data on armed robbery, the system was able to spot 10 times as many patterns as a team of experienced detectives with access to the same data.
- Selecting the most promising eggs **for in-vitro fertilization** taking into account a patient's physiology as well as results derived from different stages of an IVF treatment.
- A manager must make many different decisions when **designing a supply chain network**. Many of them involve a choice between a long-term (or less flexible) option and a short-term (or more flexible) option. If uncertainty is ignored, the long-term option will almost always be selected because it is typically cheaper. Such a decision can eventually hurt the firm, however, because actual future prices or demand may be different from what was forecasted at the time of the decision. A decision tree can be used to **evaluate decisions under uncertainty**. Combine strategic planning and financial planning during network design. Use multiple metrics to evaluate supply chain networks, financial analysis as an input to decision making, not as the decision-making process and make use of estimates along with sensitivity analysis.
- Few **other interesting applications** include- Analysis of Sudden Infant Death Syndrome; Scheduling of printed circuit board assembly lines; Analyzing amino acid sequences in Human Genome Project; Chemical material evaluation for manufacturing/production; Selecting a flight for your next travel; Identifying factors leading to better gross margins on a retail chain; Identify a strategy which would be most likely to reach a goal; Bank marketing application : Predict if the client will subscribe (yes/no) to a term deposit, by building a classification model using Decision Tree.

PART 4: CONTRIBUTION

The project was planned and implemented by all group members.

1. We all discussed the design of the project.
2. The code was built in group together with discussion and peer reviews within the group.
3. Library function was studied by each member individually and collaborated to compare and analyse the difference between our implementation of ID3 and library function.
4. Applications of decision trees was contributed by each member individually and combined together.