NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

PATTERN RECOGNITION LAB
6TH SEMESTER

SHIV KUMAR
2016UIT2563
IT-2

INDEX

1.BAYESIAN THEORY (N CLASSES, M FEATURES)

```
import pandas as pd

import numpy as np


#import data

dataset=pd.read_csv("set1.csv");

data=dataset.iloc[:,:].values

print(data)

class_arr=dataset.iloc[:,-1].values


#NO of clases

unique_class,  count_class=np.unique(class_arr,return_counts=True);

print(unique_class);

print("\n",count_class);

table_prob=[];

table_feature=[];

#------------------------------------------------------------

for i in range(data[0].size-1):

    #calculating probablity table

    # For different types of a single feature

    class_feature=data[:,i];

    print(class_feature);

    unique_feature,  count_feature=np.unique(class_feature,return_counts=True);

    print(unique_feature);

    print("\n",count_feature);

    temp=np.zeros((len(unique_feature),len(unique_class))).astype(int);

    print(temp);
```

```
    # 0---> feature YES for that class; 1----> feature NO for that class
    '''
    matrix shape would be:
      _ _ _ _ _ _ _ _ indexes of unique_class are col wise
      |
      |
      |
      indexes of unique_feature are row wise
    '''
    for j in range(len(data)):
        feature_type=data[j][i];
        class_entry=data[j][data[0].size-1];
        feature_idx, =np.where(unique_feature==feature_type);
        class_idx, =np.where(unique_class==class_entry);

        temp[feature_idx,class_idx]+=1;

    # after calculating temp matrix for feature i

    table_prob.append(temp);
    table_feature.append(unique_feature);

#--------------------------------------------------
# after finishing calculation
print("\n","\n");
print("Table Probablity","\n");
print(table_prob);

print("\n","\n");
print("Table Features","\n");
```

```
print(table_feature);


#------------------------------------
'''
Querry Solving Code from prior calculated probablities
'''


print("Enter in following order and if not to be mentioned, enter none for that feature"+"\n");
print(" CHILLS    RUNNING NOSE    HEADACHE    FEVER","\n");
print("Enter feature vector: ");
querry=[];
for i in range(data[0].size-1):
    ent=input();
    querry.append(ent);


# Class prediction code goes here
cal_prob=[]
for i in range(len(unique_class)):
    tot_prob=1;
    for j in range(len(querry)):
        if querry[j]!="none" :
            idx=np.where(table_feature[j]==querry[j]);
            count=table_prob[j][idx,i];
            tot=count_class[i];
            prob=count/tot;
            tot_prob*=prob;


    tot_prob*=(count_class[i]/np.sum(count_class));
    cal_prob.append(tot_prob);
```

```
# printing class probablities for querry and infer class to  which it belongs
max_prob=0;
max_class="";
for i in range(len(cal_prob)):
    print(unique_class[i]," : ",cal_prob[i]);
    if max_prob<cal_prob[i] :
        max_prob=cal_prob[i];
        max_class=unique_class[i];


# printing final answer
        print("\n");
print("Given feature set belongs to class ",max_class," with probablity: ",max_prob);
```

2.M-ESTIMATE

```
import pandas as pd
import numpy as np

#import data
dataset=pd.read_csv("set1.csv");
data=dataset.iloc[:,:].values
class_arr=dataset.iloc[:,-1].values

#NO of clases
unique_class,  count_class=np.unique(class_arr,return_counts=True);
print(unique_class);
print("\n",count_class);

table_prob=[];
table_feature=[];
#---------------------------------------------------------
for i in range(data[0].size-1):
    #calculating probablity table
    # For different types of a single feature
    class_feature=data[:,i];
    print(class_feature);
    unique_feature,  count_feature=np.unique(class_feature,return_counts=True);
    print(unique_feature);
    print("\n",count_feature);
    temp=np.zeros((len(unique_feature),len(unique_class))).astype(int);
    print(temp);


    # 0---> feature YES for that class; 1----> feature NO for that class
```

```
    '''
    matrix shape would be:
         _ _ _ _ _ _ _ _ indexes of unique_class are col wise
        |
        |
        |
        indexes of unique_feature are row wise
    '''
    for j in range(len(data)):
        feature_type=data[j][i];
        class_entry=data[j][data[0].size-1];
        feature_idx, =np.where(unique_feature==feature_type);
        class_idx, =np.where(unique_class==class_entry);

        temp[feature_idx,class_idx]+=1;

    # after calculating temp matrix for feature i
    table_prob.append(temp);
    table_feature.append(unique_feature);

#----------------------------------------------------
# after finishing calculation
print("\n","\n");
print("Table Probablity","\n");
print(table_prob);

print("\n","\n");
print("Table Features","\n");
print(table_feature);
```

```
#-------------------------------------
'''
Querry Solving Code from prior calculated probablities
'''

print("Enter in following order and if not to be mentioned, enter none for that feature"+"\n");
print(" CHILLS    RUNNING NOSE    HEADACHE    FEVER","\n");
print("Enter feature vector: ");
querry=[];
for i in range(data[0].size-1):
    ent=input();
    querry.append(ent);

# m-estimate calculation part
print("\n");
print("Enter m value for m-estimate: ");
m=int(input());
cal_prob=[]
for i in range(len(unique_class)):
    tot_prob=1;
    for j in range(len(querry)):
        if querry[j]!="none" :
            idx=np.where(table_feature[j]==querry[j]);
            # same as normal bayesian i.e. Pie=N(feature^class)/N(class)
            count=table_prob[j][idx,i];
            tot=count_class[i];
            prob=count/tot;
            pie=m*prob;
            # different probablity calculation for m-estimate probablity
```

```
            # N of feature
            N_feat=np.sum(table_prob[j][idx,:]);
            N_class=count_class[i];

            m_prob=(N_feat+pie)/(N_class+m);

            tot_prob*=m_prob;

    tot_prob*=(count_class[i]/np.sum(count_class));
    cal_prob.append(tot_prob);

# printing class probablities for querry and infer class to  which it belongs
max_prob=0;
max_class="";
for i in range(len(cal_prob)):
    print(unique_class[i]," : ",cal_prob[i]);
    if max_prob<cal_prob[i] :
        max_prob=cal_prob[i];
        max_class=unique_class[i];

# printing final answer
        print("\n");
print("Given feature set belongs to class ",max_class," with probablity: ",max_prob);
```

3. LOSS FUNCTION

```
import numpy as np
print("Enter number of classes", "\n");
n=int(input());
class_arr=[];
class_prob=np.zeros(n);

for i in  range(n):
    print("Enter class name","\n");
    name=input();
    print("Enter class probablity for class ",name,"\n");
    prob=float(input());
    class_arr.append(name);
    class_prob.itemset(i,prob);

print("Enter number of features: "+"\n");
no=int(input());
feature_arr=[];
for i in range(no):
    print("Enter features: ");
    name=input();
    feature_arr.append(name);

feature_prob=np.zeros((no,n));
for i in range(no):
    for j in range(n):
        print("Enter conditional probablity for feature ",feature_arr[i]," and class ", class_arr[j],"\n");
        feature_prob.itemset((i,j), float(input()));

print("Enter number of actions: "+"\n");
```

```
no=int(input());
action_arr=[];
for i in range(no):
    print("Enter action: ");
    name=input();
    action_arr.append(name);

loss_function=np.zeros((no,n));
for i in range(no):
    for j in range(n):
        print("Enter loss function for action ",action_arr[i]," and class ", class_arr[j],"\n");
        loss_function.itemset((i,j), int(input()));

# Calculation Part
tot_feature_prob=[]
for i in range(len(feature_arr)):
    temp=0;
    for j in range(n):
        temp+=feature_prob[i,j]*class_prob[j];
    tot_feature_prob.append(temp);

# likelihood probablities
table=np.zeros((n,len(feature_arr)));
for i in range(n):
    for j in range(len(feature_arr)):
        likelihood=(feature_prob[j,i]*class_prob[i])/tot_feature_prob[j];
        table.itemset((i,j),likelihood);

# Querry solving part
'''
```

OPTIMISED SOLUTION I.E. WITH MINIMUM RISK VALUE

'''

```
risk=np.zeros((len(action_arr),len(feature_arr)));
for i in range(len(action_arr)):
    for j in range(len(feature_arr)):
        temp=0;
        for k in range(n):
            temp+=table[k,j]*loss_function[i,k];
        risk.itemset((i,j),temp);


# final answer part
import sys
Min=sys.maxsize;
optimised="";
for i in range(len(risk)):
    for j in range(len(risk[0])):
        print("Risk associated with action ",action_arr[i]," and feature ",feature_arr[j]," is: ", risk[i,j]);
        if risk[i,j]<Min :
            optimised="Optimised solution is associated with action ",action_arr[i]," and feature ",feature_arr[j]," with risk value: ", risk[i,j];
            Min=risk[i,j];


print("\n");
print(optimised);
```

4. TEXT-CLASSIFICATION

```
import pandas as pd
import numpy as np

#import data
dataset=pd.read_csv("text_set.csv");
data=dataset.iloc[:,:].values
class_arr=dataset.iloc[:,-1].values

# split text line into array of characters
text_arr=[]
for  i in range(len(data)):
    string=data[i,0];
    string_arr=string.split();
    text_arr.append(string_arr);

combine=[]
for i in range(len(text_arr)):
    combine=sum([combine,text_arr[i]],[]);
print(combine);

# get unique word array
unique_text=np.unique(combine);
print("\n");
print(unique_text,"jkjkjl");
VOC=len(unique_text); # vocab value

# get unique class array
unique_class, count_class=np.unique(class_arr,return_counts=True);
print(unique_class);
```

```
#creating unique word table with their count
#creating 2d array of numbers
table=np.zeros((len(data),len(unique_text))).astype(int);
for i in range(len(data)):
    for j in range(len(unique_text)):
        count=text_arr[i].count(unique_text[j]);
        table.itemset((i,j),count);

print(table);

# array to store all count of words in a class
#total number of  words in a particular class
find_count=np.zeros(len(unique_class));
for i in range(len(text_arr)):
    number=len(text_arr[i]);
    idx=np.where(unique_class==data[i][1]);
    find_count[idx]+=number;



# probablity table

probablity_table=np.zeros((len(unique_text),len(unique_class)));
for i in range(len(unique_class)):
    # count number of words belonging to that class
    n=find_count[i];
    denom=n+VOC;
    for j in range(len(unique_text)):
        # calculating unique count of a word in a particular class
        count=0;
```

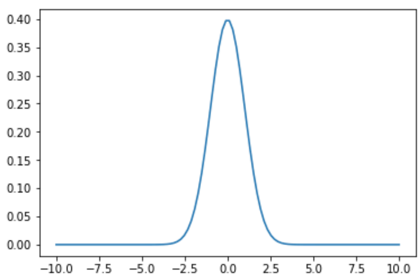
```
        for k in range(len(table)):
            if table[k][j]!=0 and class_arr[k]==unique_class[i]:
                count+=1;
        prob=(count+1)/denom;
        probablity_table.itemset((j,i),prob);

print(probablity_table);

# Classification  part
print("Enter input text for classification: ","\n");
test=input();
test_arr=test.split();

Max=0;
result="";
class_idx=0;
final_table=np.zeros(len(unique_class));
for i in range(len(unique_class)):
    tot=1;
    for j in range(len(test_arr)):
        idx=np.where(unique_text==test_arr[j]);
        tot*=probablity_table[idx,i];
    tot*=(count_class[i]/np.sum(count_class));
    final_table.itemset(i,tot);
    if tot>Max :
        Max=tot;
        class_idx=i;

for i in range(len(final_table)):
    print("\n");
```

```
    print("Probablity of belonging to class ",unique_class[i]," is: ",final_table[i]);



print("\n");

print("Text belongs to class ",unique_class[class_idx]," with probablity: ",Max);
```

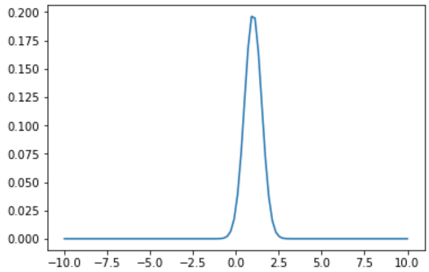
Gaussian Function for Continuous Variable

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np


# In[2]:


x = np.linspace(-10,10,100);x


# In[6]:


import numpy as np
import matplotlib.pyplot as plt

def gaussian(x,mu,sigma):

    y = np.exp(-np.power(x-mu,2)/2*np.power(sigma,2))/np.power(2*3.14*sigma*sigma,0.5)
    return y

x = np.linspace(-10,10,100)

for mu,sigma in [(0,1),(1,2),(-3,3)]:
    print()
    print('mean = ',mu,' SD = ',sigma)
    y = gaussian(x,mu,sigma)
    plt.plot(x,y)
    plt.show()
```

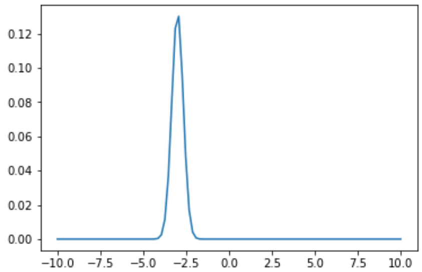
```
mean =  0  SD =  1
```



```
mean =  1  SD =  2
```




```
mean =  -3  SD =  3
```



KNN

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from collections import Counter


# In[2]:


dataset = datasets.load_breast_cancer()
X_train, X_test, Y_train, Y_test = train_test_split(dataset.data, dataset.target, test_size = 0.2,
random_state = 0)


# In[3]:


clf = KNeighborsClassifier(n_neighbors=7)
clf.fit(X_train, Y_train)


# In[4]:


clf.score(X_test, Y_test)


# In[5]:


#KNN implementation

def train(x, y):
    return

def predict_one(x_train, y_train, x_test, k):
    distances = []
    for i in range(len(x_train)):
        distance = ((x_train[i, :] - x_test)**2).sum()
        distances.append([distance, i])
    distances = sorted(distances)
    targets = []
    for i in range(k):
        index_of_training_data = distances[i][1]
        targets.append(y_train[index_of_training_data])
    return Counter(targets).most_common(1)[0][0]

def predict(x_train, y_train, x_test_data, k):
    predictions = []
    for x_test in x_test_data:
        predictions.append(predict_one(x_train, y_train, x_test, k))
```

```python
    return predictions


# In[6]:


y_pred = predict(X_train, Y_train, X_test, 7)
accuracy_score(Y_test, y_pred)
```

PCA

```
#!/usr/bin/env python
# coding: utf-8

# In[61]:


import numpy as np


# In[62]:


data = np.array([[1,2,1],[1,0,2],[1,3,1]])

data


# In[63]:


data.shape


# In[64]:


data_t = data.T
data_t


# In[65]:


cov_matrix = np.cov(data_t)
cov_matrix


# In[66]:


eigen_values,eigen_vectors = np.linalg.eig(cov_matrix)

eigen_values,eigen_vectors


# In[67]:


eig_val_vector_pair = []

for i in range(len(eigen_values)):

    eig_vec = eigen_vectors[:,i]
    eig_val_vector_pair.append((eigen_values[i],eig_vec))

eig_val_vector_pair
```

```
# In[68]:


eig_val_vector_pair.sort(reverse=True)

eig_val_vector_pair


# In[69]:


total = np.sum(eigen_values)

total


# In[77]:


k = 0
current_variance = 0

transform_matrix = []
while current_variance/total<0.99:


    current_variance+=(eig_val_vector_pair[k][0])
    k+=1
    if  k>=len(eigen_values):
        break
    print(eig_val_vector_pair[k][1])
    transform_matrix.append(eig_val_vector_pair[k][1])

transform_matrix = np.array(transform_matrix)
transform_matrix


# In[58]:


transformed_data = np.dot(data,transform_matrix.T)


# In[59]:


transformed_data
```

Decision Trees

TreeNode.py

```
import numpy as np
import pandas as pd
import math

class TreeNode:
    def __init__(self, data,output):
        # data represents the feature upon which the node was split when fitting the training data
        # data = None for leaf node
        self.data = data
        # children of a node are stored as a dicticionary with key being the value of feature upon
which the node was split
        # and the corresponding value stores the child TreeNode
        self.children = {}
        # output represents the class with current majority at this instance of the decision tree
        self.output = output
        # index will be used to assign a unique index to each node
        self.index = -1

    def add_child(self,feature_value,obj):
        self.children[feature_value] = obj
```

Classifier.py

```
import numpy as np
import pandas as pd
import math

from treenode import TreeNode
class Classifier:
    def __init__(self):
        # root represents the root node of the decision tree built after fitting the training data
        self.__root = None

    def __count_unique(self,Y):
        # returns a dictionary with keys as unique values of Y(i.e no of classes) and the
corresponding value as its frequency
        d = {}
        for i in Y:
            if i not in d:
                d[i]=1
            else:
                d[i]+=1
        return d


    def __entropy(self,Y):
        # returns the entropy
        freq_map = self.__count_unique(Y)
        entropy_ = 0
        total = len(Y)
        for i in freq_map:
            p = freq_map[i]/total
            entropy_ += (-p)*math.log2(p)
        return entropy_

    def __gain_ratio(self,X,Y,selected_feature):
```

```
        # returns the gain ratio
        info_orig = self.__entropy(Y) # info_orig represents entropy before splitting
        info_f = 0  # info_f represents entropy after splitting upon the selected feature
        split_info = 0
        values = set(X[:,selected_feature])
        df = pd.DataFrame(X)
        # Adding Y values as the last column in the dataframe
        df[df.shape[1]] = Y
        initial_size = df.shape[0]
        for i in values:
            df1 = df[df[selected_feature] == i]
            current_size = df1.shape[0]
            info_f += (current_size/initial_size)*self.__entropy(df1[df1.shape[1]-1])
            split_info += (-current_size/initial_size)*math.log2(current_size/initial_size)

        # to handle the case when split info = 0 which leads to division by 0 error
        if split_info == 0 :
            return math.inf

        info_gain = info_orig - info_f
        gain_ratio = info_gain / split_info
        return gain_ratio

    def __gini_index(self,Y):
        # returns the gini index
        freq_map = self.__count_unique(Y)
        gini_index_ = 1
        total = len(Y)
        for i in freq_map:
            p = freq_map[i]/total
            gini_index_ -= p**2
        return gini_index_

    def __gini_gain(self,X,Y,selected_feature):
        # returns the gini gain
        gini_orig = self.__gini_index(Y) # gini_orig represents gini index before splitting
        gini_split_f = 0 # gini_split_f represents gini index after splitting upon the selected feature
        values = set(X[:,selected_feature])
        df = pd.DataFrame(X)
        # Adding Y values as the last column in the dataframe
        df[df.shape[1]] = Y
        initial_size = df.shape[0]
        for i in values:
            df1 = df[df[selected_feature] == i]
            current_size = df1.shape[0]
            gini_split_f += (current_size/initial_size)*self.__gini_index(df1[df1.shape[1]-1])

        gini_gain_ = gini_orig - gini_split_f
        return gini_gain_


    def __decision_tree(self,X,Y,features,level,metric,classes):
        # returns the root of the Decision Tree(which consists of TreeNodes) built after fitting the
training data
        # Here Nodes are printed as in PREORDER traversl
        # classes represents the different classes present in the classification problem
        # metric can take value gain_ratio or gini_index
        # level represents depth of the tree
        # We split a node on a particular feature only once (in a given root to leaf node path)
```

```
# If the node consists of only 1 class
if len(set(Y)) == 1:
    print("Level",level)
    output = None
    for i in classes:
        if i in Y:
            output = i
            print("Count of",i,"=",len(Y))
        else :
            print("Count of",i,"=",0)
    if metric == "gain_ratio":
        print("Current Entropy is =  0.0")
    elif metric == "gini_index":
        print("Current Gini Index is =  0.0")

    print("Reached leaf Node")
    print()
    return TreeNode(None,output)

# If we have run out of features to split upon
# In this case we will output the class with maximum count
if len(features) == 0:
    print("Level",level)
    freq_map = self.__count_unique(Y)
    output = None
    max_count = -math.inf
    for i in classes:
        if i not in freq_map:
            print("Count of",i,"=",0)
        else :
            if freq_map[i] > max_count :
                output = i
                max_count = freq_map[i]
            print("Count of",i,"=",freq_map[i])

    if metric == "gain_ratio":
        print("Current Entropy  is =",self.__entropy(Y))
    elif metric == "gini_index":
        print("Current Gini Index is =",self.__gini_index(Y))

    print("Reached leaf Node")
    print()
    return TreeNode(None,output)


# Finding the best feature to split upon
max_gain = -math.inf
final_feature = None
for f in features :
    if metric == "gain_ratio":
        current_gain = self.__gain_ratio(X,Y,f)
    elif metric =="gini_index":
        current_gain = self.__gini_gain(X,Y,f)

    if current_gain > max_gain:
        max_gain = current_gain
        final_feature = f

print("Level",level)
```

```
        freq_map = self.__count_unique(Y)
        output = None
        max_count = -math.inf

        for i in classes:
            if i not in freq_map:
                print("Count of",i,"=",0)
            else :
                if freq_map[i] > max_count :
                    output = i
                    max_count = freq_map[i]
                print("Count of",i,"=",freq_map[i])

        if metric == "gain_ratio" :
            print("Current Entropy is =",self.__entropy(Y))
            print("Splitting on feature  X[",final_feature,"] with gain ratio ",max_gain,sep="")
            print()
        elif metric == "gini_index":
            print("Current Gini Index is =",self.__gini_index(Y))
            print("Splitting on feature  X[",final_feature,"] with gini gain ",max_gain,sep="")
            print()


        unique_values = set(X[:,final_feature]) # unique_values represents the unique values of the
feature selected
        df = pd.DataFrame(X)
        # Adding Y values as the last column in the dataframe
        df[df.shape[1]] = Y

        current_node = TreeNode(final_feature,output)

        # Now removing the selected feature from the list as we do not want to split on one feature
more than once(in a given root to leaf node path)
        index  = features.index(final_feature)
        features.remove(final_feature)
        for i in unique_values:
            # Creating a new dataframe with value of selected feature = i
            df1 = df[df[final_feature] == i]
            # Segregating the X and Y values and recursively calling on the splits
            node = self.__decision_tree(df1.iloc[:,
0:df1.shape[1]-1].values,df1.iloc[:,df1.shape[1]-1].values,features,level+1,metric,classes)
            current_node.add_child(i,node)

        # Add the removed feature
        features.insert(index,final_feature)

        return current_node

    def fit(self,X,Y,metric="gain_ratio"):
        # Fits to the given training data
        # metric can take value gain_ratio or gini_index
        features = [i for i in range(len(X[0]))]
        classes = set(Y)
        level = 0
        if metric != "gain_ratio" :
            if metric != "gini_index":
                metric="gain_ratio"  # if user entered a value which was neither gini_index nor gain_ratio
        self.__root = self.__decision_tree(X,Y,features,level,metric,classes)

    def __predict_for(self,data,node):
```

```
        # predicts the class for a given testing point and returns the answer

        # We have reached a leaf node
        if len(node.children) == 0 :
            return node.output

        val = data[node.data] # represents the value of feature on which the split was made
        if val not in node.children :
            return node.output

        # Recursively call on the splits
        return self.__predict_for(data,node.children[val])

    def predict(self,X):
        # This function returns Y predicted
        # X should be a 2-D np array
        Y = np.array([0 for i in range(len(X))])
        for i in range(len(X)):
            Y[i] = self.__predict_for(X[i],self.__root)
        return Y

    def score(self,X,Y):
        # returns the mean accuracy
        Y_pred = self.predict(X)
        count = 0
        for i in range(len(Y_pred)):
            if Y_pred[i] == Y[i]:
                count+=1
        return count/len(Y_pred)

    def export_tree_pdf(self,filename=None):
        # returns the tree as dot data
        # if filename is specified the function
        # will save the pdf file in current directory which consists of the visual reresentation of the
tree
        import pydotplus
        from collections import deque

        dot_data = '''digraph Tree {
node [shape=box] ;'''

        queue = deque()

        r = self.__root
        queue.append(r)
        count = 0
        if r.index == -1:
            r.index = count

        dot_data = dot_data + "\n{} [label=\"Feature to split upon : X[{}]\\nOutput at this node : {}
\" ];".format(count,r.data,r.output)

        # Doing LEVEL ORDER traversal in the tree (using a queue)
        while len(queue) != 0 :
            node = queue.popleft()
            for i in node.children:
                count+=1
                if(node.children[i].index==-1):
                    node.children[i].index = count
```

```
            # Creating child node
            dot_data = dot_data + "\n{} [label=\"Feature to split upon : X[{}]\\nOutput at this node :
{}\" ];".format(node.children[i].index,node.children[i].data,node.children[i].output)
            # Connecting parent node with child
            dot_data = dot_data + "\n{} -> {} [ headlabel=\"Feature value = {}\"];
".format(node.index,node.children[i].index,i)
            # Adding child node to queue
            queue.append(node.children[i])

        dot_data = dot_data + "\n}"

        if filename != None:
            graph = pydotplus.graph_from_dot_data(dot_data)
            graph.write_pdf(filename)

        return dot_data
```

In [1]:

```
import numpy as np
import pandas as pd
import math
```

In [2]:

```
from classifier import Classifier

clf1 = Classifier()
clf1
```

Out[2]:

```
<classifier.Classifier at 0x118d198d0>
```

In [3]:

```
x = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])

y = np.array([0,
              1,
              1,
              1])
```

In [4]:

```
clf1.fit(x,y)
```

```
Level 0
Count of 0 = 1
Count of 1 = 3
Current Entropy is = 0.8112781244591328
Splitting on feature  X[0] with gain ratio 0.31127812445913283

Level 1
Count of 0 = 1
Count of 1 = 1
Current Entropy is = 1.0
Splitting on feature  X[1] with gain ratio 1.0

Level 2
Count of 0 = 1
Count of 1 = 0
Current Entropy is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 1
Current Entropy is =  0.0
Reached leaf Node

Level 1
Count of 0 = 0
Count of 1 = 2
Current Entropy is =  0.0
Reached leaf Node
```

In [5]:

```
Y_pred = clf1.predict(x)
print("Predictions :",Y_pred)
print()
print("Score :",clf1.score(x,y)) # Score on training data
print()
print("DOT DATA :-",clf1.export_tree_pdf(filename="tree_OR.pdf"))
```

```
Predictions : [0 1 1 1]

Score : 1.0

DOT DATA :- digraph Tree {
node [shape=box] ;
0 [label="Feature to split upon : X[0]\nOutput at this node : 1" ];
1 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
0 -> 1 [ headlabel="Feature value = 0"];
2 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
0 -> 2 [ headlabel="Feature value = 1"];
3 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
1 -> 3 [ headlabel="Feature value = 0"];
4 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
1 -> 4 [ headlabel="Feature value = 1"];
}
```

In [7]:

```
from sklearn import datasets
# Generating a random dataset
X,Y = datasets.make_classification(n_samples=100, n_features=5, n_classes=3,n_in
formative=3 , random_state=0)

# To reduce the values a feature can take ,converting floats to int
for i in range(len(X)):
    for j in range(len(X[0])):
        X[i][j] = int(X[i][j])

clf2 = Classifier()
clf2.fit(X,Y,metric='gini_index')
Y_pred2 = clf2.predict(X)
print("Predictions : ",Y_pred2)
print()
our_score = clf2.score(X,Y)
print("Score :",our_score) # score on training data
print()
print("DOT DATA :-",clf2.export_tree_pdf(filename="tree_sample_dataset.pdf"))
```

```
Level 0
Count of 0 = 34
Count of 1 = 32
Count of 2 = 34
Current Gini Index is = 0.6663999999999999
Splitting on feature  X[4] with gini gain 0.17473071690214542

Level 1
Count of 0 = 15
Count of 1 = 19
Count of 2 = 15
Current Gini Index is = 0.6622240733027904
Splitting on feature  X[2] with gini gain 0.22894306859321423

Level 2
Count of 0 = 5
Count of 1 = 5
Count of 2 = 3
Current Gini Index is = 0.650887573964497
Splitting on feature  X[0] with gini gain 0.21499013806706113

Level 3
Count of 0 = 4
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.4444444444444444
Splitting on feature  X[1] with gini gain 0.0

Level 4
Count of 0 = 4
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.4444444444444444
Splitting on feature  X[3] with gini gain 0.0

Level 5
Count of 0 = 4
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.4444444444444444
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 2
Count of 2 = 2
Current Gini Index is = 0.5
Splitting on feature  X[1] with gini gain 0.0

Level 4
Count of 0 = 0
Count of 1 = 2
Count of 2 = 2
Current Gini Index is = 0.5
```

```
Splitting on feature  X[3] with gini gain 0.0

Level 5
Count of 0 = 0
Count of 1 = 2
Count of 2 = 2
Current Gini Index is = 0.5
Reached leaf Node

Level 3
Count of 0 = 1
Count of 1 = 0
Count of 2 = 1
Current Gini Index is = 0.5
Splitting on feature  X[1] with gini gain 0.0

Level 4
Count of 0 = 1
Count of 1 = 0
Count of 2 = 1
Current Gini Index is = 0.5
Splitting on feature  X[3] with gini gain 0.0

Level 5
Count of 0 = 1
Count of 1 = 0
Count of 2 = 1
Current Gini Index is = 0.5
Reached leaf Node

Level 2
Count of 0 = 6
Count of 1 = 6
Count of 2 = 0
Current Gini Index is = 0.5
Splitting on feature  X[3] with gini gain 0.17857142857142855

Level 3
Count of 0 = 5
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.40816326530612246
Splitting on feature  X[0] with gini gain 0.027210884353741527

Level 4
Count of 0 = 4
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.4444444444444445
Splitting on feature  X[1] with gini gain 0.0

Level 5
Count of 0 = 4
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.4444444444444445
Reached leaf Node

Level 4
Count of 0 = 1
Count of 1 = 0
```

```
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 1
Count of 1 = 1
Count of 2 = 0
Current Gini Index is = 0.5
Splitting on feature  X[1] with gini gain 0.5

Level 4
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 4
Count of 0 = 1
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 2
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 4
Count of 1 = 8
Count of 2 = 1
Current Gini Index is = 0.5207100591715975
Splitting on feature  X[1] with gini gain 0.08609467455621289

Level 3
Count of 0 = 1
Count of 1 = 6
Count of 2 = 1
Current Gini Index is = 0.40625
Splitting on feature  X[3] with gini gain 0.05625000000000002

Level 4
Count of 0 = 0
Count of 1 = 3
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 4
```

```
Count of 0 = 1
Count of 1 = 3
Count of 2 = 1
Current Gini Index is = 0.5599999999999999
Splitting on feature  X[0] with gini gain 0.0

Level 5
Count of 0 = 1
Count of 1 = 3
Count of 2 = 1
Current Gini Index is = 0.5599999999999999
Reached leaf Node

Level 3
Count of 0 = 3
Count of 1 = 2
Count of 2 = 0
Current Gini Index is = 0.48
Splitting on feature  X[0] with gini gain 0.48

Level 4
Count of 0 = 3
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 4
Count of 0 = 0
Count of 1 = 2
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 0
Count of 2 = 10
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 11
Count of 1 = 0
Count of 2 = 2
Current Gini Index is = 0.2603550295857988
Splitting on feature  X[2] with gini gain 0.15779092702169625

Level 2
Count of 0 = 5
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node
```

```
Level 2
Count of 0 = 1
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 3
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 2
Count of 1 = 0
Count of 2 = 1
Current Gini Index is = 0.4444444444444444
Splitting on feature  X[1] with gini gain 0.4444444444444444

Level 3
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 2
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 5
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 3
Count of 1 = 0
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 0
Count of 1 = 5
Count of 2 = 7
```

```
Current Gini Index is = 0.48611111111111094
Splitting on feature  X[0] with gini gain 0.34325396825396803

Level 2
Count of 0 = 0
Count of 1 = 1
Count of 2 = 6
Current Gini Index is = 0.24489795918367355
Splitting on feature  X[2] with gini gain 0.24489795918367355

Level 3
Count of 0 = 0
Count of 1 = 0
Count of 2 = 6
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 3
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Level 1
Count of 0 = 0
```

```
Count of 1 = 6
Count of 2 = 10
Current Gini Index is = 0.46875
Splitting on feature  X[1] with gini gain 0.11268939393939392

Level 2
Count of 0 = 0
Count of 1 = 3
Count of 2 = 8
Current Gini Index is = 0.39669421487603307
Splitting on feature  X[3] with gini gain 0.05578512396694213

Level 3
Count of 0 = 0
Count of 1 = 3
Count of 2 = 5
Current Gini Index is = 0.46875
Splitting on feature  X[2] with gini gain 0.04017857142857134

Level 4
Count of 0 = 0
Count of 1 = 3
Count of 2 = 4
Current Gini Index is = 0.48979591836734704
Splitting on feature  X[0] with gini gain 0.0

Level 5
Count of 0 = 0
Count of 1 = 3
Count of 2 = 4
Current Gini Index is = 0.48979591836734704
Reached leaf Node

Level 4
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 3
Count of 0 = 0
Count of 1 = 0
Count of 2 = 3
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 2
Count of 2 = 1
Current Gini Index is = 0.4444444444444445
Splitting on feature  X[2] with gini gain 0.4444444444444445

Level 3
Count of 0 = 0
Count of 1 = 2
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node
```

```
Level 3
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 0
Count of 2 = 1
Current Gini Index is =  0.0
Reached leaf Node

Level 2
Count of 0 = 0
Count of 1 = 1
Count of 2 = 0
Current Gini Index is =  0.0
Reached leaf Node

Predictions :  [0 1 1 0 1 0 2 1 2 1 1 2 1 0 1 2 1 1 1 2 2 2 0 0 2 0
0 1 1 1 0 2 0 0 2 2 2
 0 0 2 2 2 2 0 0 1 1 1 2 2 0 2 1 2 1 2 2 0 1 0 2 2 2 0 0 0 2 1 2 0 0
0 0 0
 0 0 2 0 0 2 1 0 2 1 2 2 1 1 1 0 1 0 0 1 1 0 2 0 0 0]

Score : 0.88

DOT DATA :- digraph Tree {
node [shape=box] ;
0 [label="Feature to split upon : X[4]\nOutput at this node : 0" ];
1 [label="Feature to split upon : X[2]\nOutput at this node : 1" ];
0 -> 1 [ headlabel="Feature value = 0.0"];
2 [label="Feature to split upon : X[2]\nOutput at this node : 0" ];
0 -> 2 [ headlabel="Feature value = 1.0"];
3 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
0 -> 3 [ headlabel="Feature value = 2.0"];
4 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
0 -> 4 [ headlabel="Feature value = 3.0"];
5 [label="Feature to split upon : X[0]\nOutput at this node : 2" ];
0 -> 5 [ headlabel="Feature value = -2.0"];
6 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
0 -> 6 [ headlabel="Feature value = -4.0"];
7 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
0 -> 7 [ headlabel="Feature value = -3.0"];
8 [label="Feature to split upon : X[1]\nOutput at this node : 2" ];
0 -> 8 [ headlabel="Feature value = -1.0"];
9 [label="Feature to split upon : X[0]\nOutput at this node : 0" ];
1 -> 9 [ headlabel="Feature value = 0.0"];
10 [label="Feature to split upon : X[3]\nOutput at this node : 0" ];
1 -> 10 [ headlabel="Feature value = 1.0"];
11 [label="Feature to split upon : X[1]\nOutput at this node : 1" ];
1 -> 11 [ headlabel="Feature value = 2.0"];
12 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
1 -> 12 [ headlabel="Feature value = -2.0"];
```

```
13 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
1 -> 13 [ headlabel="Feature value = -1.0"];
14 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
2 -> 14 [ headlabel="Feature value = 0.0"];
15 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
2 -> 15 [ headlabel="Feature value = 1.0"];
16 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
2 -> 16 [ headlabel="Feature value = 2.0"];
17 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
2 -> 17 [ headlabel="Feature value = -1.0"];
18 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
2 -> 18 [ headlabel="Feature value = -2.0"];
19 [label="Feature to split upon : X[2]\nOutput at this node : 2" ];
5 -> 19 [ headlabel="Feature value = 0.0"];
20 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
5 -> 20 [ headlabel="Feature value = 1.0"];
21 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
5 -> 21 [ headlabel="Feature value = -1.0"];
22 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
5 -> 22 [ headlabel="Feature value = -2.0"];
23 [label="Feature to split upon : X[3]\nOutput at this node : 2" ];
8 -> 23 [ headlabel="Feature value = 0.0"];
24 [label="Feature to split upon : X[2]\nOutput at this node : 1" ];
8 -> 24 [ headlabel="Feature value = 1.0"];
25 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
8 -> 25 [ headlabel="Feature value = 2.0"];
26 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
8 -> 26 [ headlabel="Feature value = -1.0"];
27 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
9 -> 27 [ headlabel="Feature value = 0.0"];
28 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
9 -> 28 [ headlabel="Feature value = 1.0"];
29 [label="Feature to split upon : X[1]\nOutput at this node : 1" ];
9 -> 29 [ headlabel="Feature value = -1.0"];
30 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
9 -> 30 [ headlabel="Feature value = -2.0"];
31 [label="Feature to split upon : X[0]\nOutput at this node : 0" ];
10 -> 31 [ headlabel="Feature value = 0.0"];
32 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
10 -> 32 [ headlabel="Feature value = 1.0"];
33 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
10 -> 33 [ headlabel="Feature value = -1.0"];
34 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
10 -> 34 [ headlabel="Feature value = -2.0"];
35 [label="Feature to split upon : X[3]\nOutput at this node : 1" ];
11 -> 35 [ headlabel="Feature value = 0.0"];
36 [label="Feature to split upon : X[0]\nOutput at this node : 0" ];
11 -> 36 [ headlabel="Feature value = -1.0"];
```

```
37 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
17 -> 37 [ headlabel="Feature value = 0.0"];
38 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
17 -> 38 [ headlabel="Feature value = -1.0"];
39 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
19 -> 39 [ headlabel="Feature value = 0.0"];
40 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
19 -> 40 [ headlabel="Feature value = 1.0"];
41 [label="Feature to split upon : X[2]\nOutput at this node : 2" ];
23 -> 41 [ headlabel="Feature value = 0.0"];
42 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
23 -> 42 [ headlabel="Feature value = -1.0"];
43 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
24 -> 43 [ headlabel="Feature value = 0.0"];
44 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
24 -> 44 [ headlabel="Feature value = -1.0"];
45 [label="Feature to split upon : X[3]\nOutput at this node : 0" ];
27 -> 45 [ headlabel="Feature value = 0.0"];
46 [label="Feature to split upon : X[3]\nOutput at this node : 1" ];
29 -> 46 [ headlabel="Feature value = 0.0"];
47 [label="Feature to split upon : X[3]\nOutput at this node : 0" ];
30 -> 47 [ headlabel="Feature value = -1.0"];
48 [label="Feature to split upon : X[1]\nOutput at this node : 0" ];
31 -> 48 [ headlabel="Feature value = 0.0"];
49 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
31 -> 49 [ headlabel="Feature value = -1.0"];
50 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
33 -> 50 [ headlabel="Feature value = 0.0"];
51 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
33 -> 51 [ headlabel="Feature value = -1.0"];
52 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
35 -> 52 [ headlabel="Feature value = 0.0"];
53 [label="Feature to split upon : X[0]\nOutput at this node : 1" ];
35 -> 53 [ headlabel="Feature value = -1.0"];
54 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
36 -> 54 [ headlabel="Feature value = 0.0"];
55 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
36 -> 55 [ headlabel="Feature value = -1.0"];
56 [label="Feature to split upon : X[0]\nOutput at this node : 2" ];
41 -> 56 [ headlabel="Feature value = 0.0"];
57 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
41 -> 57 [ headlabel="Feature value = -1.0"];
58 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
45 -> 58 [ headlabel="Feature value = 0.0"];
59 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
```

```
46 -> 59 [ headlabel="Feature value = -1.0"];
60 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
47 -> 60 [ headlabel="Feature value = -2.0"];
61 [label="Feature to split upon : X[None]\nOutput at this node : 0"
];
48 -> 61 [ headlabel="Feature value = 0.0"];
62 [label="Feature to split upon : X[None]\nOutput at this node : 1"
];
53 -> 62 [ headlabel="Feature value = 0.0"];
63 [label="Feature to split upon : X[None]\nOutput at this node : 2"
];
56 -> 63 [ headlabel="Feature value = 0.0"];
}
```

In [8]:

```
import sklearn.tree
clf3 = sklearn.tree.DecisionTreeClassifier()
clf3.fit(X,Y)
Y_pred3 = clf3.predict(X)
print("Predictions",Y_pred3)
sklearn_score = clf3.score(X,Y)
print("Score :",sklearn_score)
```

```
Predictions [0 1 1 0 1 0 2 1 2 1 1 2 1 0 1 2 1 1 1 2 2 2 0 0 2 0 0 1
1 1 0 2 0 0 2 2 2
 0 0 2 2 2 2 0 0 1 1 1 2 2 0 2 1 2 1 2 2 0 1 0 2 2 2 0 0 0 2 1 2 0 0
0 0 0
 0 0 2 0 0 2 1 0 2 1 2 2 1 1 1 0 1 0 0 1 1 0 2 0 0 0]
Score : 0.88
```

In [9]:

```
print("Score of our model :",our_score)
print("Score of inbuilt sklearn's decision tree on the same data :",sklearn_scor
e)
```

```
Score of our model : 0.88
Score of inbuilt sklearn's decision tree on the same data : 0.88
```

In [ ]:

```

```