In [1]:
```
import numpy as np
import pandas as pd
from numpy.linalg import inv,det
```

In [2]:
```
trainx = pd.read_csv("X-train.csv",header = None)
trainy = pd.read_csv("y-train.csv", header = None).rename(columns = {0:"y"})
testx = pd.read_csv("X-test.csv", header = None)
fulltrain = pd.concat([trainx, trainy], axis = 1)
```

In [3]:
```
#groups the data into 5 groups based on y class
def grouphelp(data):

    d = {}
    zeroes = []
    ones = []
    twos = []
    threes= []
    fours = []
    for i in range(len(data)):
        if data[i][-1] == 1:
            ones.append(data[i])
        if data[i][-1] == 2:
            twos.append(data[i])
        if data[i][-1] == 3:
            threes.append(data[i])
        if data[i][-1] == 4:
            fours.append(data[i])
        if data[i][-1] == 0:
            zeroes.append(data[i])
    d[0] = np.array(zeroes)
    d[1] = np.array(ones)
    d[2] = np.array(twos)
    d[3] = np.array(threes)
    d[4] = np.array(fours)
    return d



#finds mean and std for each feature grouped by y value
def meanstdby_y(data):
    n = len(data)
    sep = grouphelp(data)
    meanstds = {}
    pofy = {}
    for y, row in sep.items():
        meanstds[y] = meanstdhelper(row) #collects the means and std of each featur
    for i in sep:
        pofy[i] = len(sep[i])/n #collects the p(Y = y)s for each class
    return meanstds, pofy


#calculates p(x) for each feature separated by y value, for a single x vector
def gaussby_y(d,pofy, test):
    p = {}
    for y, meanstd in d.items():
        p[y] = 1
        for i in range(len(meanstd)):
            mean, std = meanstd[i]
            x = test[i]
            p[y] *= gausshelper(x, mean, std) #performing p(x1|Yi)*...*p(x19|yi)
    for yi, pyi in pofy.items():
        p[yi] = p[yi]*pyi #applying p(Y=y) to the previous term
    return p
```

```
#collects y predictions for test x vectors
def predict(d, pofy, x_test):

    l = []
    for i in range(len(x_test)):
        result = predicthelper(d, pofy, x_test[i])
        l.append(result)
    return l


#_____________________________________ helper funcs
#predicts y for a single x vector
def predicthelper(d,pofy, test):

    p = gaussby_y(d,pofy,test)
    return max(p,key=p.get)

#gaussian function calculator
def gausshelper(x, mean, std):

    denom = 1/(np.sqrt(2*np.pi)*std)
    num = np.power(np.e,(-1/2)*(x-mean)*(x-mean)/np.power(std,2))
    final = denom*num
    return final

#finds mean and std for each feature, dropping the y column
def meanstdhelper(data):

    meanstds = [(np.mean(feature), np.std(feature)) for feature in zip(*data)]
    return meanstds[:-1]
```

In [5]:
```
trainset = fulltrain.to_numpy()
info, pofy = meanstdby_y(trainset)
predictions = predict(info, pofy, trainx.to_numpy())
np.mean(predictions == trainy.to_numpy().flatten())
len(predictions)
```

Out[5]: 427

In [6]:
```
pred = predict(info,pofy,testx.to_numpy())
len(pred)
```

Out[6]: 143

In [7]:
```
np.savetxt("predictions.csv", pred, delimiter =", ")
```

EVERYTHING ABOVE THIS LINE IS PART 1

PART 2:

In [8]:
```
raind = {"temp" : [65,72,79,55,62,71,73],
"pres": [1001,1003,1030,1022,1025,1010,1011],
"rain":  [1,1,0,1,0,1,0]} #1 is yes, 0 is no
```

In [9]:
```
rain_data = pd.DataFrame(raind)
```

In [10]:
```
def gini(plus,minus):
    p = plus/(plus+minus)
    ginico = 2*p*(1-p)
    return ginico
```

In [11]:
```
parent_node = rain_data
parent_node
```

Out[11]:

	temp	pres	rain
0	65	1001	1
1	72	1003	1
2	79	1030	0
3	55	1022	1
4	62	1025	0
5	71	1010	1
6	73	1011	0

In [12]:
```
gini(4,3) #the gini coefficient of this parent node
```

Out[12]:
```
0.4897959183673469
```

for the root question: we choose "is the temp below 73?"

In [13]:
```
node11 = rain_data[rain_data["temp"] < 73] #this is the data we have in the <73 nod
node11
```

Out[13]:

	temp	pres	rain
0	65	1001	1
1	72	1003	1
3	55	1022	1
4	62	1025	0
5	71	1010	1

In [14]:
```
gini(4,1) #gini coef for node 11
```

Out[14]:
```
0.31999999999999995
```

In [15]:
```
node12 = rain_data[rain_data["temp"] >= 73] #this is the data we have in the >=73 n
node12
```

Out[15]:

	temp	pres	rain
2	79	1030	0
6	73	1011	0

In [16]:
```
gini(0,2) #gini coef for node 12
```

Out[16]:
```
0.0
```

We have gone from the initial uncertainty 0.4897959183673469 to 0 + 0.32 = 0.32, so our uncertainty has gone down

Now we have split the data into two groups. The temp>=73 group, node12 is a leaf node, the temp<73 group,node11 is an interior node.

The next question we will ask for this interior (temp<73) node is: "is the pressure below 1025?"

In [17]:
```
node21 = node11[node11["pres"] < 1025]
node21
```

Out[17]:

	temp	pres	rain
0	65	1001	1
1	72	1003	1
3	55	1022	1
5	71	1010	1

In [18]:
```
gini(4,0) #gini coef for node 21
```

Out[18]:
```
0.0
```

In [19]:
```
node22 = node11[node11["pres"]>=1025]
node22
```

Out[19]:

	temp	pres	rain
4	62	1025	0

In [20]:
```
gini(0,1) #gini coef for node 22
```

Out[20]:
```
0.0
```

We have gone from uncertainty 0.32 to 0 + 0 = 0, so our uncertainty has gone down

Now, we have our final two leaf nodes: node 21, node22. With our 3 leaf nodes, each represents the following: data points in node12 and node 22 will have a 0, or "No" for rain. Data that ends up in node 21 will have a "yes" for rain. We can express this decision tree with the following code:

In [21]:
```
#input should be a 2d array, with the following structure: [[temp1, pres1],[temp2,p
def decisiontree(data):
    yes = []
    no = []
    curr = []
    for i in data: #this loop separates the data by asking: is temp<73?
        temp = i[0]
        pres = i[1]
        if temp < 73: #if temp>73, move the data to child node
            curr.append(i)
        else: #otherwise, move data to a leaf node
            i = np.append(i,0) #adding classification
            no.append(i)

    for i in curr: #this loop separates the data by asking: is pressure < 1025?
        temp = i[0]
        pres = i[1]
        if pres < 1025:
            i = np.append(i,1) #adding classification
            yes.append(i)
        else:
            i = np.append(i,0) #adding classification
            no.append(i)
    return yes,no
```

In [22]:
```
trainx = rain_data[["temp","pres"]].to_numpy()
trainx
```

Out[22]:
```
array([[  65, 1001],
       [  72, 1003],
       [  79, 1030],
       [  55, 1022],
       [  62, 1025],
       [  71, 1010],
       [  73, 1011]], dtype=int64)
```

In [23]:
```
yes,no = decisiontree(trainx)
```

In [24]:
```
yes
```

Out[24]:
```
[array([  65, 1001,    1], dtype=int64),
 array([  72, 1003,    1], dtype=int64),
 array([  55, 1022,    1], dtype=int64),
 array([  71, 1010,    1], dtype=int64)]
```

In [25]:
```
no
```

Out[25]:
```
[array([  79, 1030,    0], dtype=int64),
 array([  73, 1011,    0], dtype=int64),
 array([  62, 1025,    0], dtype=int64)]
```

In [ ]: