1.

```
def updateWeights(vector, weights, expectedSign):
    newWeights = []
    if(expectedSign > 0):
        for i in range(len(weights)):
            newWeights.append(weights[i] + vector[i])
    else:
        for i in range(len(weights)):
            newWeights.append(weights[i] - vector[i])
    return newWeights

def dotProduct(vector1, vector2):
    total = 0
    for i in range(len(vector1)):
        total += vector1[i] * vector2[i]
    return total


positive = [[1,2,2],[1,3,5]]
negative = [[1,1,3],[1,-1,-.5]]
weight = [1,1,1]

finished = False
steps = 1
while (not finished):
    finished = True
    for vector in positive:
        dp = dotProduct(vector, weight)
        if(dp < 0):
            if(steps < 6):
                print('Update', steps , '\nvector in positive class:', vector, 'weight vector:', weight, 'dot product
with weights vector:', dp)
            weight = updateWeights(vector, weight, 1)
            finished = False
            if(steps < 6):
                print('new weights vector:', weight,'\n')
                steps+=1
    for vector in negative:
        dp = dotProduct(vector, weight)
        if(dp > 0):
            if(steps < 6):
                print('Update', steps , '\nvector in negative class:', vector, 'weight vector:', weight, 'dot product
with weights vector:', dp)
            weight = updateWeights(vector, weight, -1)
            finished = False
            if(steps < 6):
                print('new weights vector:', weight,'\n')
                steps+=1
print("Final weighted vector: ", weight)
```

```
In [8]: runfile('/home/n/HW/DataScience/HW3/q1.py', wdir='/home/n/HW/DataScience/HW3')
Update 1
vector in negative class: [1, 1, 3] weight vector: [1, 1, 1] dot product with weights vector: 5
new weights vector: [0, 0, -2]

Update 2
vector in negative class: [1, -1, -0.5] weight vector: [0, 0, -2] dot product with weights vector: 1.0
new weights vector: [-1, 1, -1.5]

Update 3
vector in positive class: [1, 2, 2] weight vector: [-1, 1, -1.5] dot product with weights vector: -2.0
new weights vector: [0, 3, 0.5]

Update 4
vector in negative class: [1, 1, 3] weight vector: [0, 3, 0.5] dot product with weights vector: 4.5
new weights vector: [-1, 2, -2.5]

Update 5
vector in positive class: [1, 2, 2] weight vector: [-1, 2, -2.5] dot product with weights vector: -2.0
new weights vector: [0, 4, -0.5]

Final weighted vector:  [0, 6, -2.5]
```

2.

```
import math

def nCr(n,r):
    f = math.factorial
    return f(n) / (f(r) * f(n-r))

def prob(n, x, p):
    return nCr(n,x) * (p ** x) * ((1 - p) ** (n-x))

def totProb(n, p):
   probability = 0
   for xVar in range(math.floor(n/2) + 1,n+1):
       probability += prob(n,xVar,p)
   return probability

print('probability for 7 learners:', totProb(7,.55),'\n')

p = 0
n = 7
while(p < .9):
    n += 1
    p = totProb(n, .55)

print('# of learners needed for 90% accuracy:', n)
```

```
In [22]: runfile('/home/n/HW/DataScience/HW3/q2.py', wdir='/home/n/HW/DataScience/HW3')
probability for 7 learners: 0.608287796875

# of learners needed for 90% accuracy: 163
```

3.

```
import numpy as np
G = np.array([[2,10],[2,5],[1,2],[4,9]])
Gmean = np.mean(G,0)

B = np.array([[8,4],[5,8],[7,5],[6,4]])
Bmean = np.mean(B,0)

S1 = np.zeros((2,2))
for j in range(len(G)):
    S1 += np.outer((G[j]-Gmean),(G[j]-Gmean))


S2 = np.zeros((2,2))
for j in range(len(B)):
    S2 += np.outer((B[j]-Bmean),(B[j]-Bmean))
SW = S1+S2

from numpy.linalg import inv
SW_inv = inv(SW)
W = np.matmul(SW_inv,(Gmean-Bmean))## Weight vector for projection


Gmean_proj = np.matmul(W.T,Gmean)
Bmean_proj = np.matmul(W.T,Bmean)
zcut = 0.5 *(Gmean_proj + Bmean_proj)
print(Gmean_proj,Bmean_proj,zcut)
test = np.matmul(W.T,[3,3])
print(test)
if(test>zcut):
    print('[3,3] is in class1')
else:
    print('[3,3] is in class2')
```

```
In [31]: runfile('/home/n/HW/DataScience/HW3/untitled2.py', wdir='/home/n/HW/DataScience/HW3')
-0.647594985288474 -2.6790008954842017 -1.663297940386338
-1.201995650505309
[3,3] is in class1
```

4.

```
def calcGini(table,colIndex):
    attributeDict = {}
```

```
    for row in table:
        if row[colIndex] not in attributeDict:
            attributeDict[row[colIndex]] = [1,0]
        else:
            attributeDict[row[colIndex]][0] += 1
        if row[3] == '+':
            attributeDict[row[colIndex]][1] += 1
    gi = 0
    for key in attributeDict:
        weight = attributeDict[key][0]
        numPos = attributeDict[key][1]
        gi += (weight/6) * (1 - ((numPos/weight)**2 + ((weight-numPos)/weight)**2))
    # print(attributeDict)
    return gi


x = [['red', 'square', 'big', '+'],['blue','square','big', '+'],['red','round','small','-'],['green','square','small','-'],
['red','round','big','+'],['green','square','big','-']]
colDict = {0:'Color', 1:'Shape', 2:'Size'}
for col in range(3):
    gi = calcGini(x, col)
    if col == 0:
        bestgi = gi
        best = colDict[col]
    elif gi < bestgi:
        estgi = gi
        best = colDict[col]
    print('GI for col', colDict[col]+': ', gi)
print('root node attribute:', best)
```

```
In [55]: runfile('/home/n/HW/DataScience/HW3/untitled3.py', wdir='/home/n/HW/DataScience/HW3')
GI for col Color:  0.2222222222222222
GI for col Shape:  0.5
GI for col Size:  0.25
root node attribute: Color
```

5.

```
from math import e
def sigmoid(x):
    return 1 / (1 +(e**(x * -1)))
x0, x1, x2= 1,1,0
u3 = sigmoid(x0 + 3 * x1 + 4 * x2)
u4 = sigmoid(-6 * x0 + 6 * x1 + 5 * x2)
u5 = sigmoid(-3.93 * x0 + 2 * u3 + 4 * u4)
print('output: ', u5)
```

```
In [64]: runfile('/home/n/HW/Data
output:  0.5085060742863741
```

I couldn’t figure out the second part of question 5