Assign 1

November 6, 2020

0.0.1 Question 1:

Generate 1000 1D samples from a normally distributed random variable (with arbitrary mu, and sigma) and compute its sample, mean and variance.

```
1. See that the sample mean and variance is a reasonably good estimate
2. The unbiased variance estimate that we have learned is a better estimate than
   the sample variance
3. Also see that for large number of samples, all these estimates are going closer
   and closer to the true estimate.
```

[2]:
```python
import numpy as np

data = np.random.randn(1000,1)
```

[3]:
```python
print("The mean is ", np.mean(data))
```

```
The mean is  0.002593066476275599
```

[4]:
```python
print("The standard deviation is ", np.std(data))
```

```
The standard deviation is  1.0296679801094437
```

[5]:
```python
# Particular Mean

New_mean = 120
New_std = 12

data_new = New_mean + New_std*(np.random.randn(1000,1))
```

[6]:
```python
print("The new mean is ", np.mean(data_new))
```

```
The new mean is  120.22814302793869
```

[7]:
```python
print("The new standard deviation is ", np.std(data_new))
```

```
The new standard deviation is  12.155276474517468
```

Part 1

[8]:
```python
# Sample mean calculation
sample_m = np.sum(data_new)/len(data_new)
```

```
print("The sample mean is ",sample_m)
```

```
The sample mean is  120.22814302793869
```

We can see that the sample mean is a resonably good estimate of True mean.

[9]:
```
#Sample Variance Calculation
sample_v = np.sum((data_new - np.mean(data_new))**2)/len(data_new)
print("The sample variance" ,sample_v)

# Since Sigma or Standard Deviation is the root of variance
sigma = np.sqrt(sample_v)
print("The sample standard deviation is ",sigma)
```

```
The sample variance 147.75074617195781
The sample standard deviation is  12.155276474517468
```

We can see that sample standard deviation is a resonably good estimate of True standard deviation.

Part 2

[10]:
```
#Biased variance sample estimate
sample_v = np.sum((data_new - np.mean(data_new))**2)/len(data_new)
print("The sample Biased standard deviation is ", np.sqrt(sample_v))

#Unbiased variance sample estimate
sample_un = np.sum((data_new - np.mean(data_new))**2)/(len(data_new)-1)
print("The sample Unbiased standard deviation is ", np.sqrt(sample_un))
```

```
The sample Biased standard deviation is  12.155276474517468
The sample Unbiased standard deviation is  12.161358674785255
```

Part 3

[13]:
```
# Particular Mean

New_mean = 120
New_std = 12

data_new = New_mean + New_std*(np.random.randn(1000,1))
```

[14]:
```
print("The new mean is ", np.mean(data_new))
```

```
The new mean is  120.66053136112913
```

We can see that even though our True mean is 120, our calculated sample mean is only giving a resonably good estimate. Since we know that Sample mean is an UNBIASED estimate if we take large samples then the sample mean should converge on the True mean.

[23]:
```
data_new1 = New_mean + New_std*(np.random.randn(100000,1))
print("The new mean is ", np.mean(data_new1))
```

```
The new mean is  120.00347546082845
```

[25]:
```
data_new2 = New_mean + New_std*(np.random.randn(10000000,1))
print("The new mean is ", np.mean(data_new2))
```

```
The new mean is  120.00640650251303
```

We can see that it is getting closer to the true mean.

[32]:
```
data_new1 = New_mean + New_std*(np.random.randn(100000,1))
sample_un1 = np.sum((data_new1 - np.mean(data_new1))**2)/(len(data_new1)-1)
print("The sample Unbiased standard deviation is ", np.sqrt(sample_un1))
```

```
The sample Unbiased standard deviation is  12.032060168934715
```

We can see that Unbiased sample variance is also converging on the True Standard Variance.

[ ]:
```

```

Assign 2

November 6, 2020

0.0.1 Question 2:

Write a program to implement expectation maximization algorithm to separate the samples from coin A and coin B that we have seen in the Lecture.

[1]:
```
import numpy as np
Coin = np.
↪array([[1,0,0,0,1,1,0,1,0,1],[1,1,1,1,0,1,1,1,1,1],[1,0,1,1,1,1,1,0,1,1,],[1,0,1,0,0,0,1,1,
Coin
```

[1]:
```
array([[1, 0, 0, 0, 1, 1, 0, 1, 0, 1],
       [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
       [1, 0, 1, 1, 1, 1, 1, 0, 1, 1],
       [1, 0, 1, 0, 0, 0, 1, 1, 0, 0],
       [0, 1, 1, 1, 0, 1, 1, 1, 0, 1]])
```

[2]:
```
def counterhead(A):
    Ar=A.tolist()
    r=0
    for i in range(len(Ar)):
        if Ar[i]==1:
            r+=1
    return r
```

[3]:
```
#k=counterhead(Coin[3,:])
```

[4]:
```
#Using Binomial Distribution with the parametric form
def binomial_likelihood(thet1,thet2,Arr):
    k=counterhead(Arr)
    n=len(Arr)
    Like_CoinA=(thet1**k)*((1-thet1)**(n-k))
    Like_CoinB=(thet2**k)*((1-thet2)**(n-k))
    return Like_CoinA,Like_CoinB,k,n
```

[5]:
```
#x,y,c,q=binomial_likelihood(0.4,0.8,Coin[0,:])
#x,y,c,q
```

[6]:
```python
def new_theta(array,theta_A,theta_B):


    LikeA,LikeB,k,n=binomial_likelihood(theta_A,theta_B,array)

    Total_try=LikeA+LikeB
    N_Like_CoinA=LikeA/Total_try
    N_Like_CoinB=LikeB/Total_try
    #print(N_Like_CoinA,N_Like_CoinB)

    Sucess_A=N_Like_CoinA*k
    Sucess_B=N_Like_CoinB*k
    Fail_A=N_Like_CoinA*(n-k)
    Fail_B=N_Like_CoinB*(n-k)


    return Sucess_A,Sucess_B,Fail_A,Fail_B
```

[8]:
```python
#w,q,r,s = new_theta(Coin[4,:],0.4,0.8)
#w,q,r,s
```

[9]:
```python
def driver(main_array,thet_A,thet_B):
    su_a=[]
    su_b=[]
    fa_a=[]
    fa_b=[]
    for i in range(5):
        array_in=main_array[i,:]

        t,y,u,i=new_theta(array_in,thet_A,thet_B)
        #print(t,y,u,i)

        su_a.append(t)
        su_b.append(y)
        fa_a.append(u)
        fa_b.append(i)
        #print(su_a,su_b)

    return su_a,su_b,fa_a,fa_b
```

[10]:
```python
#Testing function
SuccessA,SucessB,FailA,FailB = driver(Coin,0.4,0.8)
#SuccessA,SucessB,FailA,FailB
```

[11]:
```
def theta_update(tet1,tet2,S_A,S_B,F_A,F_B):
    tet1=sum(S_A)/(sum(S_A)+sum(F_A))
    tet2=sum(S_B)/(sum(S_B)+sum(F_B))
    return tet1,tet2
```

[12]:
```
#u,i=theta_update(0.4,0.8,SuccessA,SucessB,FailA,FailB)
#u,i
```

[63]:
```
def main_function(theta_main1,theta_main2,array_main):
    theta1_array=[]
    theta2_array=[]

    for j in range(150):


        SuccessA,SucessB,FailA,FailB =␣
 ↪driver(array_main,theta_main1,theta_main2)
            ␣
 ↪tet1_out,tet2_out=theta_update(theta_main1,theta_main2,SuccessA,SucessB,FailA,FailB)
        theta_main1=tet1_out
        theta_main2=tet2_out

        theta1_array.append(tet1_out)
        theta2_array.append(tet2_out)


        if(j>0):
            if((theta1_array[j]-theta1_array[j-1]<10**-10) and␣
 ↪(theta1_array[j]-theta1_array[j-1]<10**-10)):
                break




    #print(theta2_array,theta1_array)
    return tet1_out,tet2_out
```

Intialise with a random theta values as theta1 and theta2

[73]:
```
theta_final1,theta_final2=main_function(0.8,0.2,Coin)
print("The final theta1 and theta2 values are ",theta_final1,theta_final2)
```

```
The final theta1 and theta2 values are  0.7967890668593683 0.5195831198837334
```

[ ]: