Q4

May 19, 2019

```
In [1]: import numpy as np
        import pandas as pd
        from scipy.linalg import sqrtm
```

0.1 Calculate the pmf and Derivatives

FIrst we can make the pmf of y

$$p_y(y) = \pi \cdot I(y=0) + (1-\pi)\frac{e^{-\lambda}\lambda^y}{y!}$$

Let $\theta = (\lambda, \pi)$ and $Y = (y_1, \ldots, y_n)$ Then likelihood is,

$$\begin{aligned} L(\theta|Y) &= \prod_{i=1}^{n}\left[\pi \cdot I(y_i=0) + (1-\pi)\frac{e^{-\lambda}\lambda^{y_i}}{y_i!}\right] \\ &= \prod_{y_i=0}\left[\pi + (1-\pi)\frac{e^{-\lambda}\lambda^0}{0!}\right]\prod_{y_i\neq 0}\left[(1-\pi)\frac{e^{-\lambda}\lambda^{y_i}}{y_i!}\right] \end{aligned}$$

$N = \sum_{k=0}^{6} n_k$, Then log likelihood $l(\theta)$ is

$$\begin{aligned} l(\theta) &= \sum_{y_i=0} log\left[\pi + (1-\pi)\frac{e^{-\lambda}\lambda^0}{0!}\right] + \sum_{y_i\neq 0} log\left[(1-\pi)\frac{e^{-\lambda}\lambda^{y_i}}{y_i!}\right] \\ &= n_0 log\left[\pi + (1-\pi)e^{-\lambda}\right] + (N-n_0)\left[log(1-\pi) - \lambda\right] + \sum_{y_i\neq 0}\left[y_i log\lambda - log(y_i!)\right] \end{aligned}$$

Then, log likelihood function is

```
In [2]: def loglikelihood(lam,pi,Y):
            n0 = sum(Y==0)
            N = len(Y)
            out = n0 * np.log(pi + (1-pi)*np.exp(-lam)) \
                  + (N-n0)*(np.log(1-pi) - lam) \
                  +(Y*np.log(lam)\
                  - np.array(list(map(np.math.factorial,Y)))).sum()
            return(out)
```

First derivative is

$$
\begin{aligned}
\frac{\partial l(\theta)}{\partial \lambda} &= (-n_0)\cdot\frac{(1-\pi)e^{-\lambda}}{\pi+(1-\pi)e^{-\lambda}} - (N-n_0) + \sum_{y_i\neq 0}\left(\frac{y_i}{\lambda}\right)\\
&= (-n_0)\cdot\frac{(1-\pi)e^{-\lambda}}{\pi+(1-\pi)e^{-\lambda}} - (N-n_0) + \sum_{y_i}\left(\frac{y_i}{\lambda}\right)\\
\frac{\partial l(\theta)}{\partial \pi} &= n_0\cdot\frac{1-e^{-\lambda}}{\pi+(1-\pi)e^{-\lambda}} - (N-n_o)\cdot\frac{1}{1-\pi}
\end{aligned}
$$

Second Derivatice is

$$
\begin{aligned}
\frac{\partial^2 l\left(\theta\right)}{\partial \lambda^2} &= n_0\cdot\frac{\pi\left(1-\pi\right)e^{-\lambda}}{\left(\pi+\left(1-\pi\right)e^{-\lambda}\right)^2} - \sum_{y_i\neq 0}\frac{y_i}{\lambda^2}\\
&= n_0\cdot\frac{\pi\left(1-\pi\right)e^{-\lambda}}{\left(\pi+\left(1-\pi\right)e^{-\lambda}\right)^2} - \sum_{y_i}\frac{y_i}{\lambda^2}\\
\frac{\partial^2 l\left(\theta\right)}{\partial \pi^2} &= (-n_0)\cdot\frac{\left(1-e^{-\lambda}\right)^2}{\left(\pi+\left(1-\pi\right)e^{-\lambda}\right)^2} - (N-n_0)\,\frac{1}{\left(1-\pi\right)^2}\\
\frac{\partial^2 l\left(\theta\right)}{\partial \pi\partial\lambda} &= n_0\cdot\frac{e^{-\lambda}}{\left(\pi+\left(1-\pi\right)e^{-\lambda}\right)^2}
\end{aligned}
$$

1 (a) Derive Newton's Method and Fisher Scoring Method

1.1 Newton's method

Iterate

$$\theta^{(t+1)} = \theta^{(t)} - \left[l''(\theta^{(t)})\right]^{-1} l'(\theta^{(t)})$$

and Standard Error Estimate is

$$\sqrt{\left(-l''(\theta)\right)^{-1}}$$

We calculated $l'(\theta)$ and $l''(\theta)$, first make the dataset and derivative functions

```
In [3]: Y = np.concatenate([np.repeat(0,3062),np.repeat(1,587),np.repeat(2,284),
                            np.repeat(3,103),np.repeat(4,33),np.repeat(5,4),
                            np.repeat(6,2)])
        Y.shape
```

```
Out[3]: (4075,)
```

```
In [4]: def first_der_lam(lam,pi,y):
            n0 = sum(y==0)
            N = len(y)
            out = -n0 *(((1-pi)*np.exp(-lam))/(pi+(1-pi)*np.exp(-lam))) \
                  -(N-n0) + y.sum()/lam
            return(out)
```

```
In [5]: def first_der_pi(lam,pi,y):
            n0 = sum(y==0)
            N = len(y)
            out = n0*((1-np.exp(-lam))/(pi+(1-pi)*np.exp(-lam))) \
                  -(N-n0)/(1-pi)
            return(out)
```

```
In [6]: def second_der_lam2(lam,pi,y):
            n0 = sum(y==0)
            N = len(y)
            out = n0*((pi*(1-pi)*np.exp(-lam))/((pi + (1-pi)*np.exp(-lam))**2))\
                  -y.sum()/(lam**2)
            return(out)
```

```
In [7]: def second_der_pi2(lam,pi,y):
            n0 = sum(y==0)
            N = len(y)
            out = -n0*(((1-np.exp(-lam))**2)/((pi-(1-pi)*np.exp(-lam))**2)) \
                  -(N-n0)/((1-pi)**2)
            return(out)
```

```
In [8]: def second_der_pilam(lam,pi,y):
            n0 = sum(y==0)
            N = len(y)
            out = n0*((np.exp(-lam))/((pi+(1-pi)*np.exp(-lam))**2))
            return(out)
```

Set the initail value

$$\lambda_0 = 1$$
$$\pi_0 = 0.5$$

```
In [9]: lam , pi = 1, 0.5
        theta = np.array([lam,pi])
```

Make the function iterate until loglikelihood do not increase more than criteria

```
In [10]: def Newton(theta,Y,citeria = 10**(-7)):
             llikelst = [loglikelihood(theta[0],theta[1],Y)]
             thetalst = [theta]
             niter = 0
             while True:
                 niter = niter + 1
                 l1 = np.array([first_der_lam(theta[0],theta[1],Y)
                                ,first_der_pi(theta[0],theta[1],Y)])
                 l2 = np.reshape([second_der_lam2(theta[0],theta[1],Y),
                              second_der_pilam(theta[0],theta[1],Y),
                              second_der_pilam(theta[0],theta[1],Y),
                              second_der_pi2(theta[0],theta[1],Y)],(2,2))
                 theta = theta - np.linalg.inv(l2).dot(l1)
```

```
            thetalst.append(theta)
            llikelst.append(loglikelihood(theta[0],theta[1],Y))
            if (abs(llikelst[-1]-llikelst[-2]) < citeria):
                break
        out = pd.DataFrame({'lambda' : pd.DataFrame(thetalst)[0],
                            'pi': pd.DataFrame(thetalst)[1],
                            'logLikelihood':llikelst})
        stdm = sqrtm(-np.linalg.inv(l2))
        return(out,stdm)
```

```
In [11]: N_result,N_stdm = Newton(theta,Y)
```

Then the result is

$$\hat{\theta}^{Newton} = (\hat{\lambda}^{Newton}, \hat{\pi}^{Newton}) = (1.037836, 0.615055)$$

it converges at 13 times

```
In [12]: N_result
```

Out[12]:

	lambda	pi	logLikelihood
0	1.000000	0.500000	-10425.367474
1	0.866135	0.529549	-10396.762382
2	0.939669	0.562936	-10386.699016
3	0.990509	0.589546	-10381.765834
4	1.018312	0.604416	-10380.371074
5	1.030320	0.610933	-10380.109878
6	1.035009	0.613502	-10380.069499
7	1.036783	0.614476	-10380.063694
8	1.037446	0.614841	-10380.062881
9	1.037693	0.614976	-10380.062768
10	1.037785	0.615027	-10380.062752
11	1.037819	0.615046	-10380.062750
12	1.037832	0.615053	-10380.062750
13	1.037836	0.615055	-10380.062750

Standard error estimate is

```
In [13]: pd.DataFrame(N_stdm)
```

Out[13]:

	0	1
0	0.036015	0.004470
1	0.004470	0.009601

1.2 Fisher Scoring method

Iterate

$$\theta^{(t+1)} = \theta^{(t)} + \left[I(\theta^{(t)})\right]^{-1} l'(\theta^{(t)})$$

where $I(\theta) = E\left[-l''(\theta)\right]$, and Standard Error Estimate is

$$\sqrt{\left[I(\theta^{(t)})\right]^{-1}}$$

We calculated $l'(\theta)$, $l''(\theta)$, we only need to calculate $I(\theta) = E\left[-l''(\theta)\right]$

Since

$$n_0, \ldots, n_6 \sim Multinomial\left(N, \left\{\pi + (1-\pi)e^{-\lambda}, (1-\pi)\frac{\lambda^1 e^{-\lambda}}{1!}, \ldots, \frac{\lambda^6 e^{-\lambda}}{6!}\right\}\right)$$

Expected value of n_k is

$$E[n_0] = N \cdot (\pi + (1-\pi)e^{-\lambda})$$
$$E[n_k] = N \cdot \left((1-\pi)\frac{\lambda^k e^{-\lambda}}{k!}\right) \text{ for i} = 1, \ldots .6$$

Then $I(\theta) = E\left[-l''(\theta)\right]$ is

$$\begin{aligned}
E\left[-\frac{\partial^2 l(\theta)}{\partial \lambda^2}\right] &= -E[n_0] \cdot \frac{\pi(1-\pi)e^{-\lambda}}{\left(\pi + (1-\pi)e^{-\lambda}\right)^2} + E\left[\sum_{y_i \neq 0} \frac{y_i}{\lambda^2}\right] \\
&= -E[n_0] \cdot \frac{\pi(1-\pi)e^{-\lambda}}{\left(\pi + (1-\pi)e^{-\lambda}\right)^2} + E\left[\sum_{k=1}^{6} \frac{k \cdot n_k}{\lambda^2}\right] \\
&= -E[n_0] \cdot \frac{\pi(1-\pi)e^{-\lambda}}{\left(\pi + (1-\pi)e^{-\lambda}\right)^2} + \sum_{k=1}^{6} \frac{k \cdot E[n_k]}{\lambda^2} \\
E\left[-\frac{\partial^2 l(\theta)}{\partial \pi^2}\right] &= E[n_0] \cdot \frac{\left(1-e^{-\lambda}\right)^2}{\left(\pi + (1-\pi)e^{-\lambda}\right)^2} - (N - E[n_0])\frac{1}{(1-\pi)^2} \\
E\left[-\frac{\partial^2 l(\theta)}{\partial \pi \partial \lambda}\right] &= -E[n_0] \cdot \frac{e^{-\lambda}}{\left(\pi + (1-\pi)e^{-\lambda}\right)^2}
\end{aligned}$$

Make the function of expectation of n_k's where $k \neq 0$

```
In [14]: def E_nk(N,k,lam,pi):
             return (N*(1-pi)*(lam**k)*np.exp(-lam))/(np.math.factorial(k))
```

```
In [15]: def E_second_der_lam2(lam,pi,y):
             N = len(y)
             n0 = N*(pi + (1-pi)*np.exp(-lam))
             summ = 0
             for k in range(7):
                 summ  = summ + k*E_nk(N,k,lam,pi)
             out = n0*((pi*(1-pi)*np.exp(-lam))/((pi + (1-pi)*np.exp(-lam))**2))\
                 -(summ)/(lam**2)
             return(out)
```

```
In [16]: def E_second_der_pi2(lam,pi,y):
             N = len(y)
             n0 = N*(pi + (1-pi)*np.exp(-lam))
             out = -n0*(((1-np.exp(-lam))**2)/((pi-(1-pi)*np.exp(-lam))**2)) \
                 -(N-n0)/((1-pi)**2)
             return(out)
```

In [17]:
```python
def E_second_der_pilam(lam,pi,y):
    N = len(y)
    n0 = N*(pi + (1-pi)*np.exp(-lam))
    out = n0*((np.exp(-lam))/((pi+(1-pi)*np.exp(-lam))**2))
    return(out)
```

Set the initail value

$$\lambda_0 = 1$$
$$\pi_0 = 0.5$$

In [18]:
```python
lam , pi = 1, 0.5
theta = np.array([lam,pi])
```

In [19]:
```python
def Fisher(theta,Y,citeria = 10**(-7)):
    llikelst = [loglikelihood(theta[0],theta[1],Y)]
    thetalst = [theta]
    niter = 0
    while True:
        niter = niter + 1
        l1 = np.array([first_der_lam(theta[0],theta[1],Y)
                      ,first_der_pi(theta[0],theta[1],Y)])
        l2 = np.reshape([E_second_der_lam2(theta[0],theta[1],Y),
                     E_second_der_pilam(theta[0],theta[1],Y),
                     E_second_der_pilam(theta[0],theta[1],Y),
                     E_second_der_pi2(theta[0],theta[1],Y)],(2,2))
        theta = theta - np.linalg.inv(l2).dot(l1)
        thetalst.append(theta)
        llikelst.append(loglikelihood(theta[0],theta[1],Y))
        if (abs(llikelst[-1]-llikelst[-2]) < citeria):
            break
    out = pd.DataFrame({'lambda' : pd.DataFrame(thetalst)[0],
                      'pi': pd.DataFrame(thetalst)[1],
                      'logLikelihood':llikelst})
    stdm = sqrtm(-np.linalg.inv(l2))
    return(out,stdm)
```

In [20]:
```python
F_result,F_stdm = Fisher(theta,Y)
```

Then the result is

$$\hat{\theta}^{Fisher} = (\hat{\lambda}^{Fisher}, \hat{\pi}^{Fisher}) = (1.037836, 0.615055)$$

it converges at 13 times

In [21]:
```python
F_result
```

Out[21]:

	lambda	pi	logLikelihood
0	1.000000	0.500000	-10425.367474
1	0.915897	0.538016	-10393.735425

```
2    0.950009   0.568840   -10385.365635
3    0.992221   0.591847   -10381.488303
4    1.018426   0.605006   -10380.339002
5    1.030359   0.611067   -10380.106903
6    1.035041   0.613536   -10380.069205
7    1.036799   0.614487   -10380.063659
8    1.037453   0.614844   -10380.062876
9    1.037696   0.614978   -10380.062767
10   1.037786   0.615027   -10380.062752
11   1.037819   0.615046   -10380.062750
12   1.037832   0.615053   -10380.062750
13   1.037836   0.615055   -10380.062750
```

Standard error estimates is

```
In [22]: pd.DataFrame(F_stdm)
```

```
Out[22]:          0          1
         0  0.036041   0.004474
         1  0.004474   0.009601
```

2 (b) EM algorithn

Let Z be random varible

$$Z \sim Bernoulli(\pi)$$

and $\theta = (\lambda, \pi)$ Then,

$$\begin{aligned} L(\theta|Y,Z) &= P(Y,Z|\theta) \\ &= P(Y|Z,\theta)P(Z|\theta) \\ &= \prod_i I(y_i = 0)^{z_i} \left(\frac{e^{-\lambda}\lambda^{y_i}}{y_i!}\right)^{1-z_i} \prod_i \pi^{z_i}(1-\pi)^{1-z_i} \\ &= \prod_i [\pi \cdot I(y_i = 0)]^{z_i} \left[(1-\pi)\left(\frac{e^{-\lambda}\lambda^{y_i}}{y_i!}\right)\right]^{1-z_i} \end{aligned}$$

log likelihood is

$$l(\theta|Y,Z) = \sum_i [z_i log(\pi \cdot I(y_i = 0)) + (1-z_i)\{log(1-\pi) - \lambda + y_i log(\lambda) - log(y_i!)\}]$$

2.0.1 E-step

$$\begin{aligned} Q(\theta|\theta^{(t)}) &= E[l(\theta|Y_{com})|Y_{obs}, \theta^{(t)}] \\ &= \sum_i \left[E[z_i|Y,\theta^{(t)}] log(\pi \cdot I(y_i = 0)) + (1 - E[z_i|Y,\theta^{(t)}])\{log(1-\pi) - \lambda + y_i log(\lambda) - log(y_i!)\}\right] \end{aligned}$$

Where

$$
\begin{aligned}
E[z_i|Y,\theta^{(t)}] &= P(z_i = 1|Y,\theta^{(t)}) \\
&= \frac{P(y_i|z_i = 1,\theta^{(t)})P(z_i = 1)}{P(y_i|z_i = 0,\theta^{(t)})P(z_i = 0) + P(y_i|z_i = 1,\theta^{(t)})P(z_i = 1)} \\
&= \begin{cases} 0 & \text{when } y_i \neq 0 \\ \frac{\pi^{(t)}}{\pi^{(t)}+(1-\pi^{(t)})e^{-\lambda^{(t)}}} & \text{when } y_i = 0 \end{cases}
\end{aligned}
$$

2.0.2 M-step

First partial derivative for λ,

$$
\begin{aligned}
&\frac{\partial Q(\theta|\theta^{(t)})}{\partial \lambda} =^{let} 0 \\
\rightarrow &\sum_i (1 - E[z_i|Y,\theta^{(t)}])(-1 + \frac{y_i}{\lambda}) = 0 \\
\rightarrow &\lambda^{(t+1)} = \frac{\sum_i (1 - E[z_i|Y,\theta^{(t)}]) \cdot y_i}{\sum_i (1 - E[z_i|Y,\theta^{(t)}])}
\end{aligned}
$$

Second partial derivative for π,

$$
\begin{aligned}
&\frac{\partial Q(\theta|\theta^{(t)})}{\partial \pi} =^{let} 0 \\
\rightarrow &\sum_i \frac{E[z_i|Y,\theta^{(t)}]}{\pi} - \sum_i \frac{1 - E[z_i|Y,\theta^{(t)}]}{1-\pi} = 0 \\
\rightarrow &\pi^{(t+1)} = \frac{1}{N}\sum_i E[z_i|Y,\theta^{(t)}]
\end{aligned}
$$

```
In [23]: def Estep(lam,pi,Y):
             if Y ==0:
                 out = pi/(pi + (1-pi)*np.exp(-lam))
             else:
                 out = 0
             return(out)
```

```
In [24]: def Mstep(lam,pi,Y):
             lam = sum(list(map(lambda y : (1-Estep(lam,pi,y))*y,Y)))\
                   /sum(list(map(lambda y : (1-Estep(lam,pi,y)),Y)))
             pi = sum(list(map(lambda y : Estep(lam,pi,y),Y)))/len(Y)
             return(lam,pi)
```

```
In [25]: def iterateEM(theta,Y,citeria = 10**(-7)):
             llikelst = [loglikelihood(theta[0],theta[1],Y)]
             thetalst = [theta]
             while True:
                 theta = Mstep(theta[0],theta[1],Y)
                 thetalst.append(theta)
```

```
            llikelst.append(loglikelihood(theta[0],theta[1],Y))
            if (abs(llikelst[-1]-llikelst[-2]) < citeria):
                break
        out = pd.DataFrame({'lambda' : pd.DataFrame(thetalst)[0],
                            'pi': pd.DataFrame(thetalst)[1],
                            'logLikelihood':llikelst})
        return(out)
```

Set the initail value

$$\lambda_0 = 1$$
$$\pi_0 = 0.5$$

In [26]:
```
lam , pi = 1, 0.5
theta = np.array([lam,pi])
```

In [27]:
```
EM_result = iterateEM(theta,Y)
```

In [28]:
```
EM_result
```

Out[28]:

	lambda	pi	logLikelihood
0	1.000000	0.500000	-10425.367474
1	0.886469	0.532120	-10395.553509
2	0.890872	0.552198	-10390.295073
3	0.915914	0.567316	-10386.553374
4	0.941596	0.579144	-10383.965771
5	0.963787	0.588331	-10382.322126
6	0.981853	0.595353	-10381.333153
7	0.996074	0.600637	-10380.760983
8	1.007004	0.604565	-10380.439774
9	1.015254	0.607456	-10380.263591
10	1.021396	0.609568	-10380.168651
11	1.025921	0.611102	-10380.118167
12	1.029230	0.612212	-10380.091588
13	1.031635	0.613013	-10380.077695
14	1.033376	0.613590	-10380.070472
15	1.034633	0.614005	-10380.066731
16	1.035538	0.614303	-10380.064799
17	1.036188	0.614516	-10380.063804
18	1.036656	0.614669	-10380.063291
19	1.036991	0.614779	-10380.063028
20	1.037231	0.614858	-10380.062892
21	1.037404	0.614914	-10380.062823
22	1.037527	0.614955	-10380.062787
23	1.037616	0.614984	-10380.062769
24	1.037679	0.615004	-10380.062760
25	1.037724	0.615019	-10380.062755
26	1.037757	0.615030	-10380.062752
27	1.037780	0.615038	-10380.062751
28	1.037797	0.615043	-10380.062750

```
29  1.037809  0.615047  -10380.062750
30  1.037818  0.615050  -10380.062750
31  1.037824  0.615052  -10380.062750
```

Then the result is

$$\hat{\theta}^{EM} = (\hat{\lambda}^{EM}, \hat{\pi}^{EM}) = (1.037824, 0.615052)$$

it converges at 31 times

3 (c) compare the result

Newton's method result is

$$\hat{\theta}^{Fisher} = (\hat{\lambda}^{Fisher}, \hat{\pi}^{Fisher}) = (1.037836, 0.615055)$$

it converges at 13 times

Fisher scoring method result is

$$\hat{\theta}^{Fisher} = (\hat{\lambda}^{Fisher}, \hat{\pi}^{Fisher}) = (1.037836, 0.615055)$$

it converges at 13 times

EM result is

$$\hat{\theta}^{EM} = (\hat{\lambda}^{EM}, \hat{\pi}^{EM}) = (1.037824, 0.615052)$$

it converges at 31 times

Estimated value of θ is very similar but EM algorithm converges slower than Newton's and Fisher scoring method