Machine Learning I at TU Berlin

Assignment 5 - Group PTHGL

November 19, 2018

Exercise 1

Discrete EM: Coin Tosses from Multiple Distributions

$$P(Z = z_i|\theta) = \begin{cases} \lambda & \text{if } z_i = heads \\ 1-\lambda & \text{if } z_i = tails \end{cases}$$

$$P(\mathcal{X} = x|Z = z_i, \theta) = \begin{cases} p_1^{h(x_i)}(1-p_1)^{t(x_i)} & \text{if } z_i = heads \\ p_2^{h(x_i)}(1-p_2)^{t(x_i)} & \text{if } z_i = tails \end{cases}$$

where $h(x_i)$ is the number of heads and $t(x_i)$ is the number of tails. Therefore, the joint probability is:

$$\begin{aligned} P(\mathcal{X} = x, Z = z|\theta) &= \prod_{i=1}^{N} P(Z = z_i|\theta)P(\mathcal{X} = x|Z = z_i, \theta) \\ &= \prod_{i=1}^{N} \lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)} \end{aligned}$$

Furthermore, $P(Z = z_i|\mathcal{X} = x, \theta^{old})$ is given by:

$$R_1(i) = \frac{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

and

$$R_2(i) = \frac{(1-\lambda) p_2^{h(x_i)}(1-p_2)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

Then we get:

$$\begin{aligned} Q(\theta, \theta^{old}) &= \sum_{i=1}^{N} P(Z = z|\mathcal{X} = x, \theta^{old}) log[P(\mathcal{X} = x, Z = z|\theta)] \\ &= \sum_{i=1}^{N} R_1(i) log[\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + R_2(i) log[p_2^{h(x_i)}(1-p_2)^{t(x_i)}] \end{aligned}$$

Maximizing this function leads to $\theta^{new} = (\lambda', p_1', p_2')$:

$$\begin{aligned} \frac{\partial Q}{\partial \lambda} &= \sum_{i=1}^{N} \frac{R_1(i)}{\lambda} - \frac{1}{(1-\lambda)} + \frac{R_1(i)}{(1-\lambda)} = 0 \\ \therefore & \sum_{i=1}^{N} R_1(i) - N\lambda = 0 \\ \Rightarrow & \lambda' = \frac{\sum R_1(i)}{N}. \end{aligned}$$

$$\begin{aligned} \frac{\partial Q}{\partial p_1'} &= \frac{\partial}{\partial p_1'} \sum_{i=1}^{N} \{R_1(i)[log(p_1^{h(x_i)}) + log((1-p_1)^{t(x_i)})] + R_2(i)[log(p_2^{h(x_i)}) + log((1-p_2)^{t(x_i)})]\} = 0 \\ \therefore & \sum_{i=1}^{N} \{h(x_i)R_1(i) - h(x_i)R_1(i)p_1 - t(x_i)R_1(i)p_1\} = 0 \\ \Rightarrow & p_1' = \frac{\sum_{i=1}^{N} h(x_i)R_1(i)}{M \sum_{i=1}^{N} R_1(i)}. \end{aligned}$$

$$\begin{aligned} \frac{\partial Q}{\partial p_2'} &= \frac{\partial}{\partial p_2'} \sum_{i=1}^{N} \{R_1(i)[log(p_1^{h(x_i)}) + log((1-p_1)^{t(x_i)})] + R_2(i)[log(p_2^{h(x_i)}) + log((1-p_2)^{t(x_i)})]\} = 0 \\ \therefore & \sum_{i=1}^{N} \{h(x_i)R_2(i) - h(x_i)R_2(i)p_2 - t(x_i)R_2(i)p_2\} = 0 \\ \Rightarrow & p_2' = \frac{\sum_{i=1}^{N} h(x_i)R_2(i)}{M \sum_{i=1}^{N} R_2(i)}. \end{aligned}$$

sheet05

November 19, 2018

1 Expectation-Maximization

In this assignment we will be using the Expectation Maximization method to estimate the parameters of the same three coin experiment as in the theoretical part. We will examine the behavior of the algorithm for various combinations of parameters.

1.1 Description of the Experiment

The following procedure generates the data for the three coin experiment.
The parameters are:

- λ := The probability of heads on the hidden coin H.
- p_1 := The probability of heads on coin A.
- p_2 := The probability of heads on coin B.

Each of the N samples is collected the following way:

- The secret coin (H) is tossed.
- If the result is heads, coin A is tossed M times and the results are recorded.
- If the result is tails, coin B is tossed M times and the results are recorded.

Heads are recorded as 1.
Tails are recorded as 0.
The data is returned as an $N \times M$ matrix, where each of the N rows correspond to the trials and contains the results of the corresponding sample (generated either by coin A or by coin B).

1.2 Description of Provided Functions

Three functions are provided for your convenience:

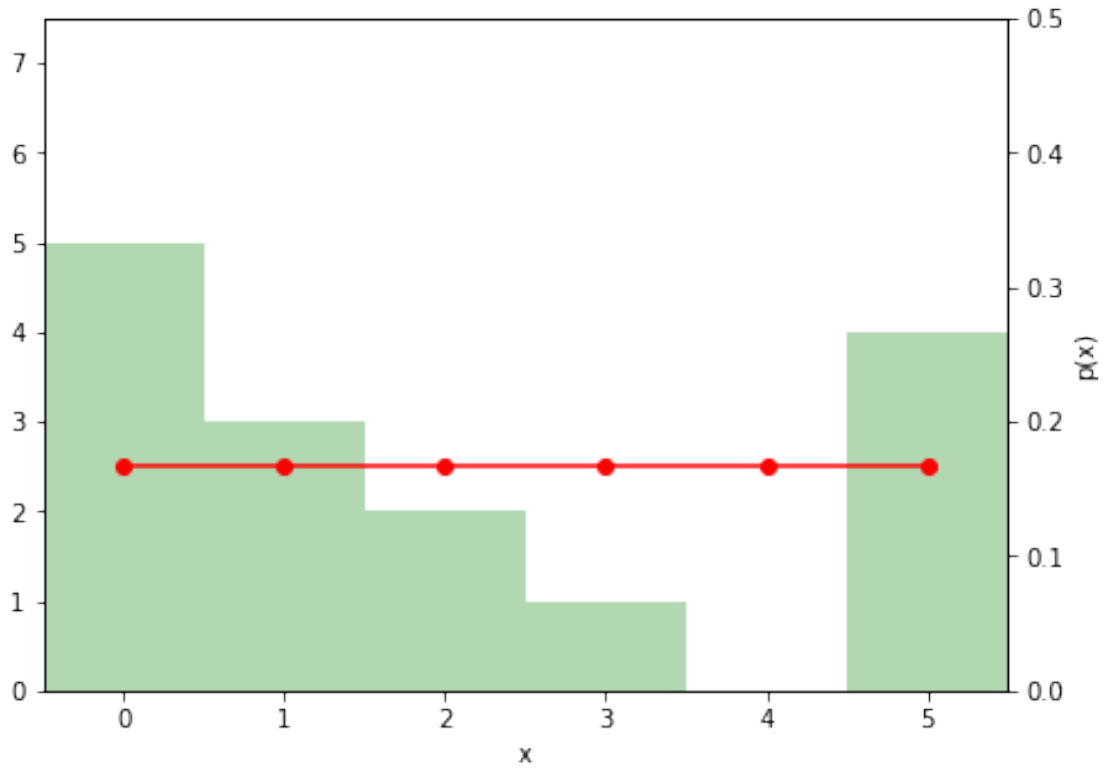
- `utils.generateData(lambda,p1,p2,N,M)`: Performs the experiment N times with coin parameters specified as argument and returns the results in a $N \times M$ matrix.
- `utils.unknownData()` Returns a dataset of size $N \times M$ where generation parameters are unknown.

- `utils.plot(data,distribution)`: Plot a histogram of the number of heads per trial along with the probability distribution. This function will be used to visualize the progress of the EM algorithm at every iteration.

An example of use of these two functions is given below:

```
In [17]: %matplotlib inline
         import numpy,utils

         # Print the data matrix as a result of the three coins experiment with parameter 0.5, 0
         data = utils.generateData(0.5,0.8,0.2,15,5)
         print(data)

         # Print the data histogram along with a uniform probability distribution.
         utils.plot(data,numpy.ones([data.shape[1]+1])/(data.shape[1]+1))
```

```
[[0 0 0 0 0]
 [0 0 0 1 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [0 0 0 0 0]
 [0 1 1 0 0]
 [0 0 0 0 1]
 [1 0 0 1 0]
 [1 1 1 1 1]
 [0 0 0 0 0]
 [0 1 0 1 1]
 [0 0 0 0 0]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```



1.3 Calculate the Log-Likelihood (10 P)

Implement a function which calculates the log likelihood for a given dataset and parameters. The log-likelihood is given by:

$$LL = \frac{1}{N}\sum_{i=1}^{N}\log \sum_{z\in\{\text{heads,tails}\}} P(X = x_i, Z = z \mid \theta) = \frac{1}{N}\sum_{i=1}^{N}\log\left[\lambda \cdot p_1^{h(x_i)} \cdot (1-p_1)^{t(x_i)} + (1-\lambda)\cdot p_2^{h(x_i)} \cdot (1-p_2)^{t(x_i)}\right]$$

where $h(x_i)$ and $t(x_i)$ denote the number of heads and tails in sample i, respectively. Note that we take the averaged log-likelihood over all trials, hence the multiplicative term $\frac{1}{N}$ in front.

```
In [18]: import numpy as np

         data = utils.unknownData()

         def loglikelihood(data,lam,p1,p2):

             N = data.shape[0]
             M = data.shape[1]

             h = np.sum(data,axis = 1)
             t = data.shape[1]-h
```

```
    a = np.log(lam*p1**h * (1-p1)**t + (1-lam)*p2**h*(1-p2)**t)

    LL = np.sum(a)/N

    return LL
```

1.4 Implementing and Running the EM Algorithm (30 P)

Implement a function which iteratively determines the values of λ, p_1 and p_2. The function starts with some initial estimates for the parameters and returns the results of the method for those parameters.

In each iteration, the following update rules are used for the parameters:

$$\lambda^{new} = \frac{E(\#heads(coin\_H))}{\#throws(coin\_H)} = \frac{1}{N}\sum_{i=1}^{N} \frac{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

$$p_1^{new} = \frac{E(\#heads(coin\_A))}{E(\#throws(coin\_A))} = \frac{\sum_{i=1}^{N} R_1(i)h(x_i)}{M\sum_{i=1}^{N} R_1(i)}$$

$$p_2^{new} = \frac{E(\#heads(coin\_B))}{E(\#throws(coin\_B))} = \frac{\sum_{i=1}^{N} R_2(i)h(x_i)}{M\sum_{i=1}^{N} R_2(i)}$$

where $h(x_i)$ and $t(x_i)$ denote the number of heads and tails in sample i, respectively, and

$$R_1(i) = \frac{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

$$R_2(i) = \frac{(1-\lambda) p_2^{h(x_i)}(1-p_2)^{t(x_i)}}{\lambda p_1^{h(x_i)}(1-p_1)^{t(x_i)} + (1-\lambda)p_2^{h(x_i)}(1-p_2)^{t(x_i)}}$$

TODO:

- **Implement the EM learning procedure.**
- **Use as stopping criterion the improvement of log-likelihood between two iterations to be smaller than** 0.001**.**
- **Run the EM procedure on the data returned by function** `utils.unknownData()`**. Use as an initial solution for your model the parameters** $\lambda = 0.5$**,** $p_1 = 0.25$**,** $p_2 = 0.75$ **.**
- **At each iteration of the EM procedure, print the log-likelihood and the value of your model parameters, and plot the learned probability distribution using the function utils.plot().**

```
In [19]: import utils
         %matplotlib inline
         import scipy.stats as stats

         def EM(data,lam,p1,p2,show_all_plots):
```

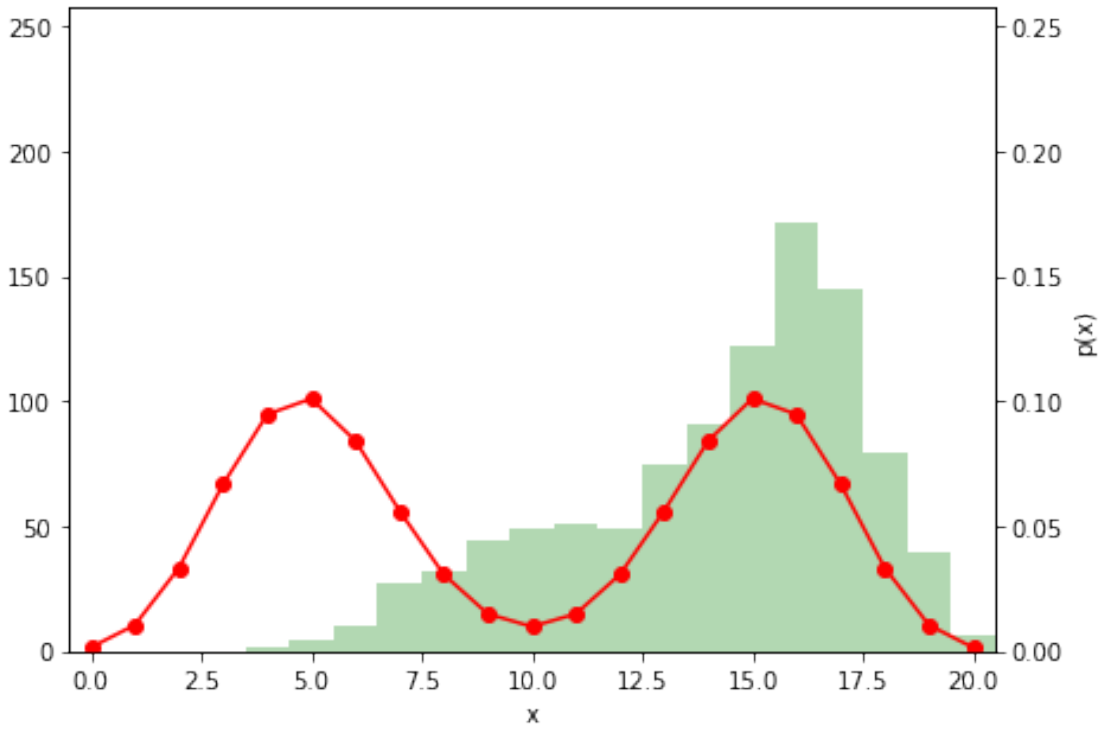
```
# define fix values N,h,t
N = data.shape[0]
M = data.shape[1]
h = np.sum(data,axis = 1)
t = data.shape[1]-h

criterion = False # (to be set to True)

#get current loglikelihood and set to old
LL = loglikelihood(data,lam,p1,p2)

#set counter to 0
it = 0

#show starting result
print('it:%2d  lambda: %.2f  p1: %.2f  p2: %.2f  log-likelihood: %.3f'%(it, lam, p1

#get the distribution, by calculating two binomial distributions and adding them
#pdf of heads on coinA
distA = lam*stats.binom.pmf(range(M+1),data.shape[1],p1)
distB = (1-lam)*stats.binom.pmf(range(M+1),data.shape[1],p2)

#show plot if wanted
if(show_all_plots):
    utils.plot(data,distA+distB)

#define some helpful functions
def newLam(lam,p1,p2):

    newLam = np.sum((lam*p1**h*(1-p1)**t) / (lam*p1**h*(1-p1)**t + (1-lam)*p2**h*(1
    return newLam

def newP1(lam,p1,p2):

    R1 = (lam*p1**h*(1-p1)**t) / (lam*p1**h*(1-p1)**t + (1-lam)*p2**h*(1-p2)**t)
    newP1 = np.sum(R1*h)/np.sum(M*R1)
    return newP1


def newP2(lam,p1,p2):

    R2 =  ((1-lam)*p2**h*(1-p2)**t) / (lam*p1**h*(1-p1)**t + (1-lam)*p2**h*(1-p2)**
    newP2 = np.sum(R2*h)/np.sum(M*R2)
    return newP2


# Iterate until the stopping criterion is satisfied
```

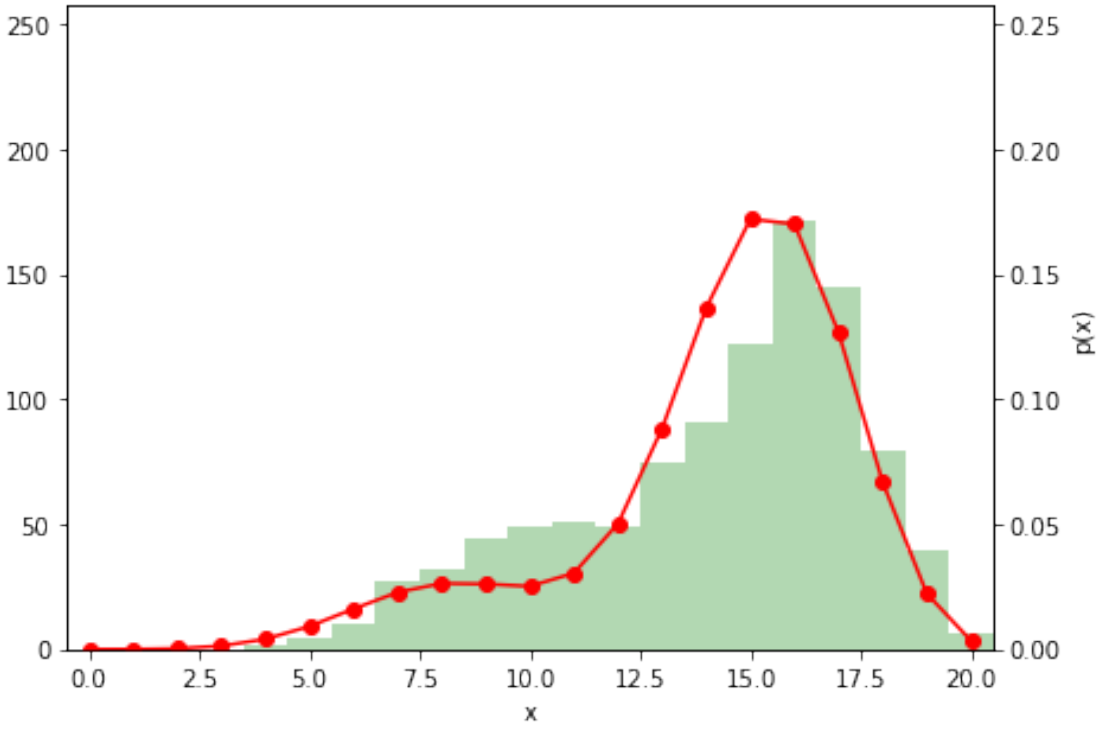
```
        while (criterion == False):

            #count up iterator
            it = it+1

            #store old likelihood
            oldLL = LL

            #store old values for calculation of new values
            oldLam = lam
            oldP1 = p1
            oldP2 = p2

            #get new values for lam,p1 and p2
            lam = newLam(oldLam,oldP1,oldP2)
            p1 = newP1(oldLam,oldP1,oldP2)
            p2 = newP2(oldLam,oldP1,oldP2)

            #calc loglikelihoodd
            LL = loglikelihood(data,lam,p1,p2)

            #print result
            print('it:%2d  lambda: %.2f  p1: %.2f  p2: %.2f  log-likelihood: %.3f'%(it, lam

            distA = lam*stats.binom.pmf(range(M+1),data.shape[1],p1)
            distB = (1-lam)*stats.binom.pmf(range(M+1),data.shape[1],p2)

            #plot result
            if(show_all_plots):
                utils.plot(data,distA+distB)


            #check if criterion has been met
            if(np.abs(oldLL-LL) < 0.001):
                criterion = True

                #only show final plot if before wasnt shown
                if(not show_all_plots):
                    utils.plot(data,distA+distB)

                print("EM has converged")

    # ----------------------------------------
```

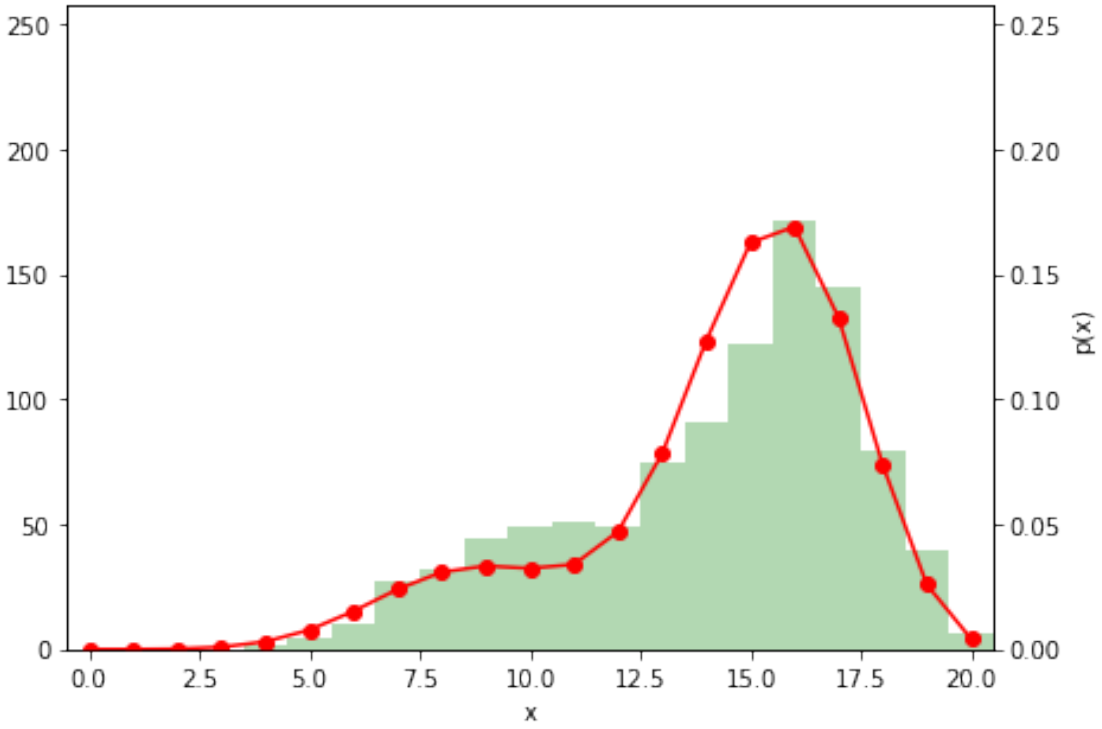
```
In [20]: EM(utils.unknownData(),0.5,0.25,0.75,True)
```

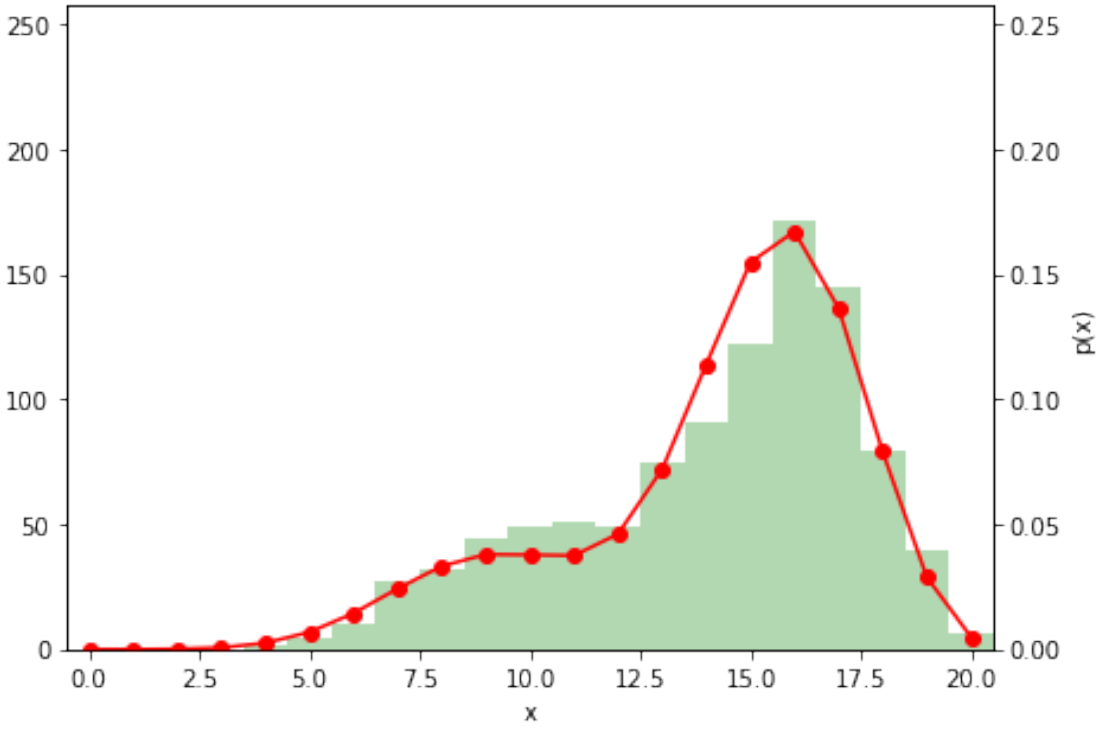
```
it: 0  lambda: 0.50  p1: 0.25  p2: 0.75  log-likelihood: -12.201
it: 1  lambda: 0.15  p1: 0.41  p2: 0.76  log-likelihood: -11.709
```

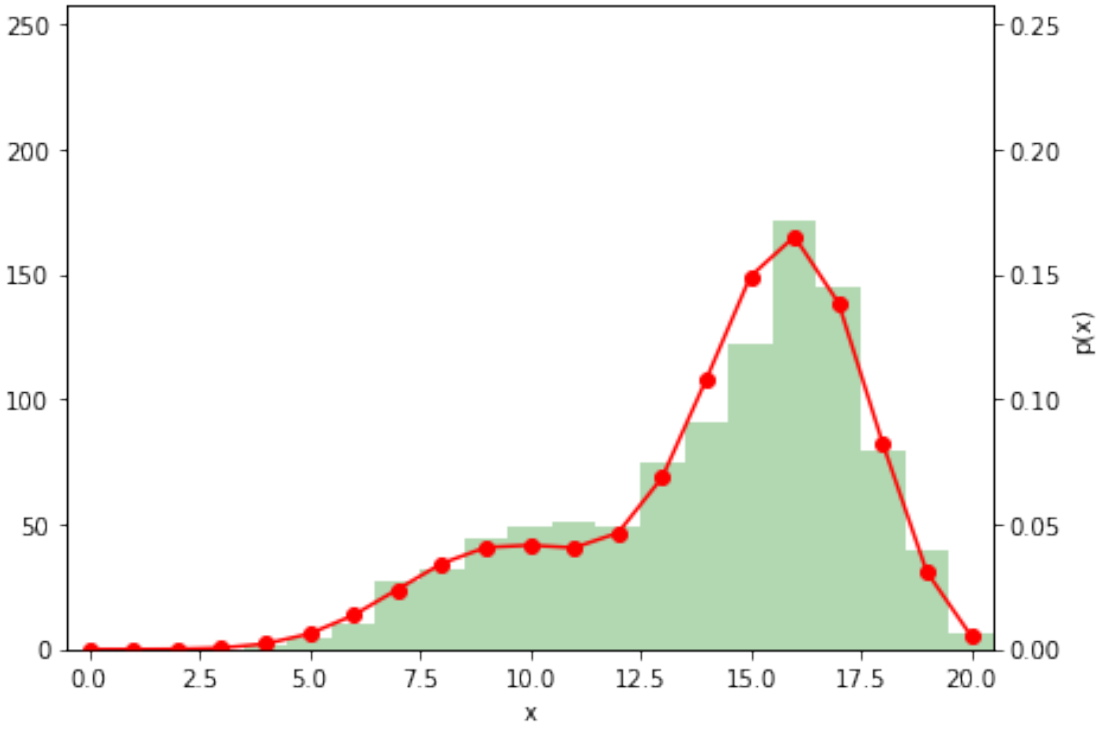
```
it: 2  lambda: 0.18  p1: 0.44  p2: 0.77  log-likelihood: -11.684
it: 3  lambda: 0.21  p1: 0.46  p2: 0.78  log-likelihood: -11.672
it: 4  lambda: 0.23  p1: 0.47  p2: 0.78  log-likelihood: -11.666
it: 5  lambda: 0.24  p1: 0.48  p2: 0.78  log-likelihood: -11.664
it: 6  lambda: 0.25  p1: 0.48  p2: 0.79  log-likelihood: -11.662
it: 7  lambda: 0.26  p1: 0.48  p2: 0.79  log-likelihood: -11.662
EM has converged
```

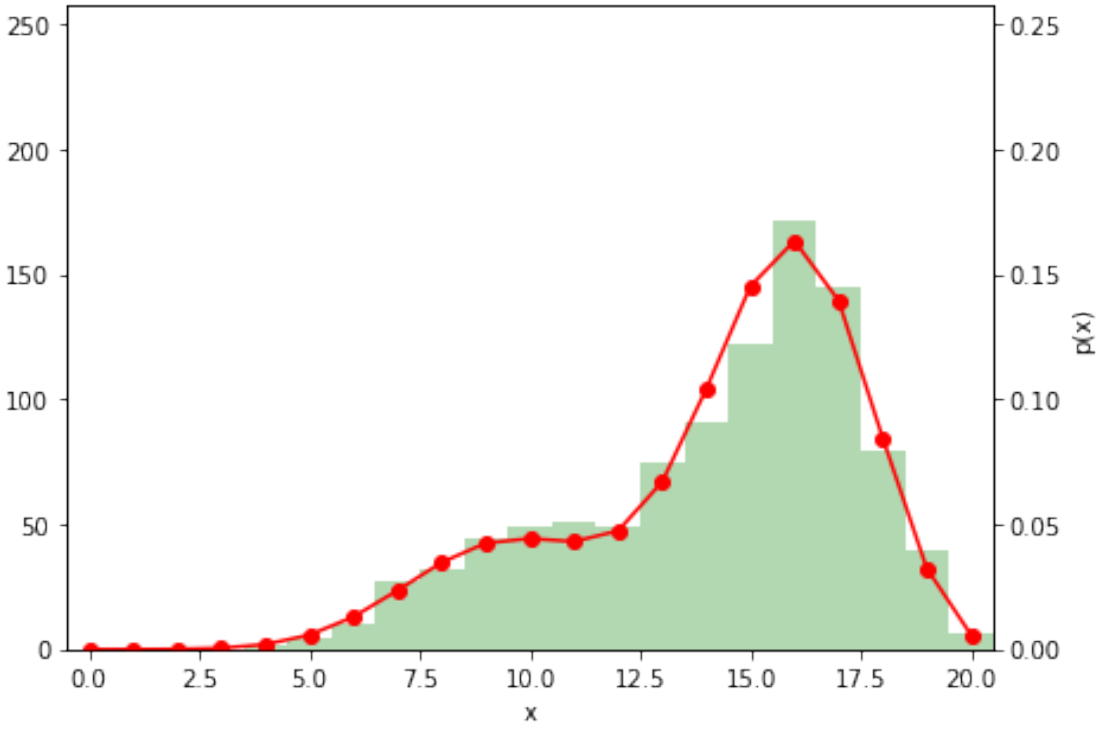


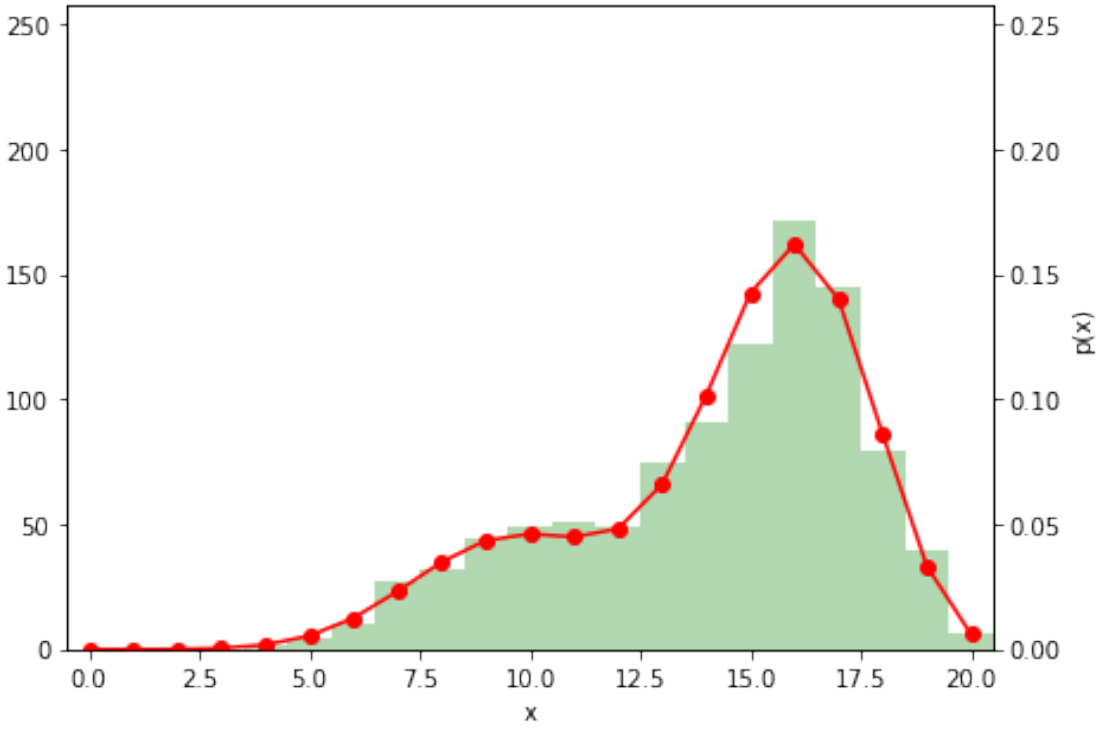

250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x

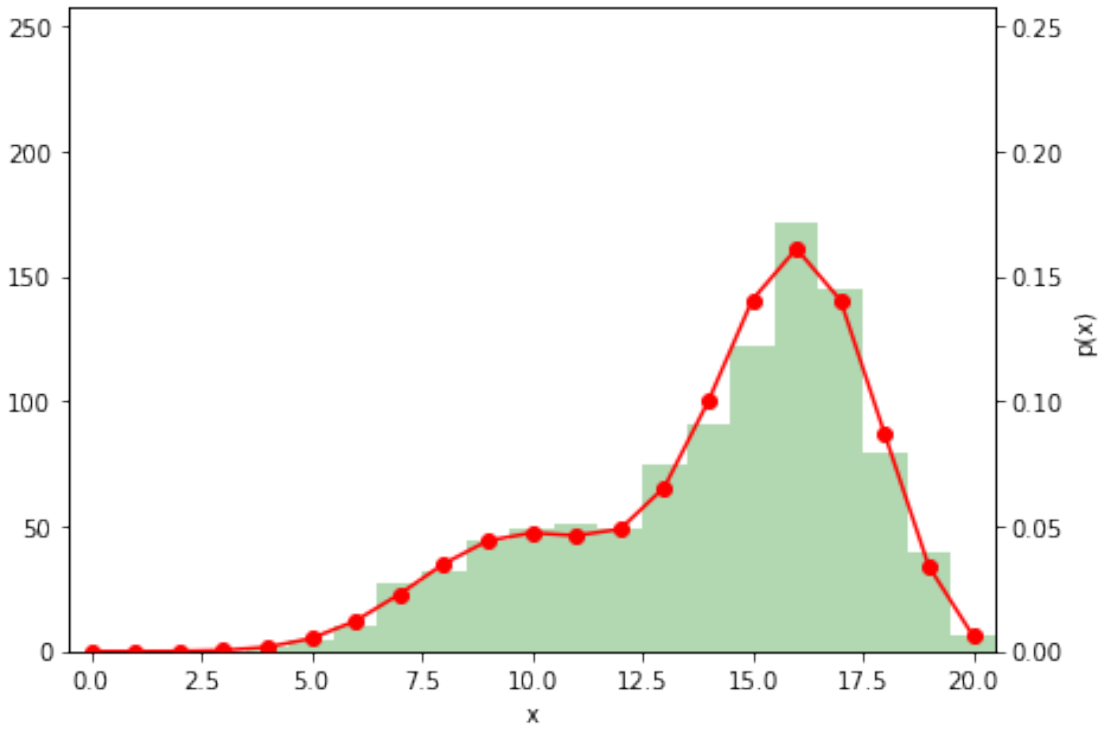

250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x


250
200
150
100
50
0
0.25
0.20
0.15
0.10
0.05
0.00
p(x)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
x



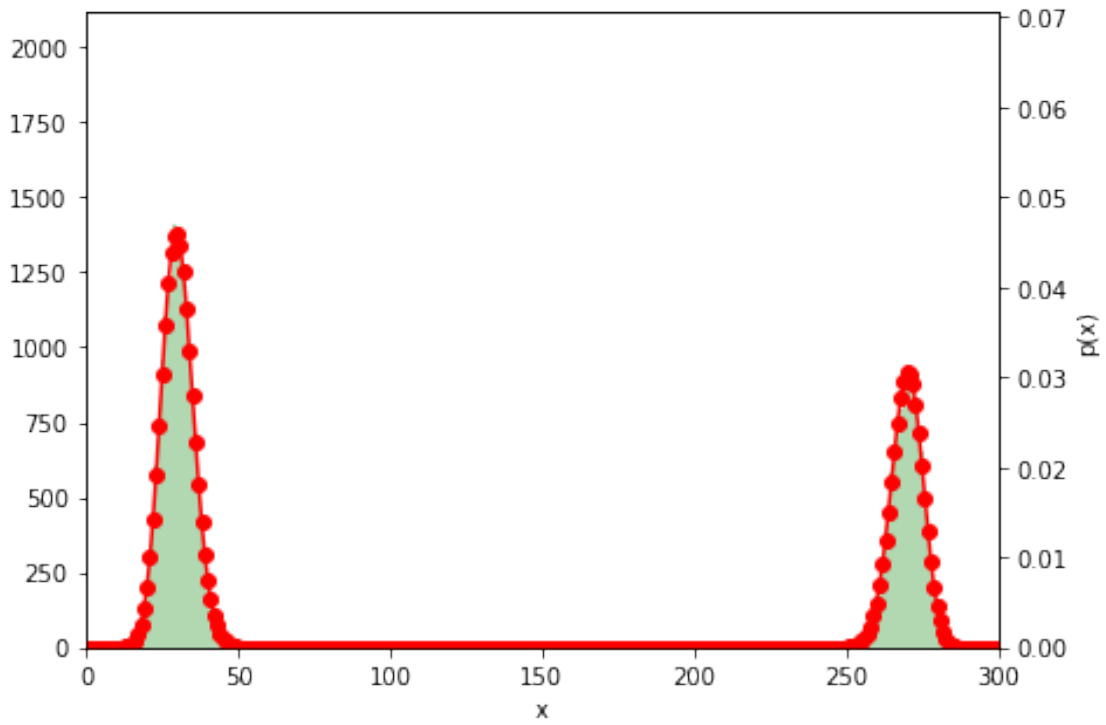
1.5 More Experiments (10 P)

Examine the behaviour of the EM algorithm for various combinations of data generation parameters and initializations (for generating various distributions, use the method `utils.generateData(...)`). In particular, find settings for which:

- The role of coins A and B are permuted between the data generating model and the learned model (i.e. $\hat{p}_1 \approx p_2$, $\hat{p}_2 \approx p_1$ and $\hat{\lambda} \approx 1 - \lambda$).
- The EM procedure takes a long time to converge.

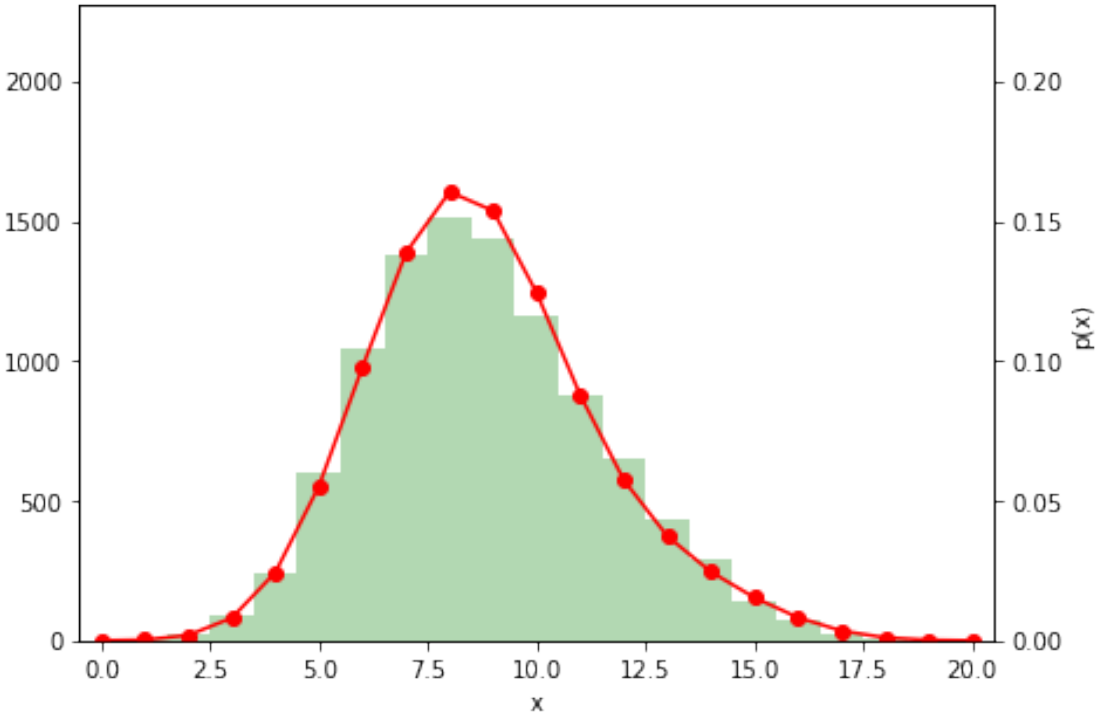
Print the parameters and log-likelihood objective at each iteration. Only display the plot for the converged model.

```
In [21]: EM(utils.generateData(0.6,0.1,0.9,30000,300),0.4,0.7,0.3,False)

it: 0  lambda: 0.40  p1: 0.70  p2: 0.30  log-likelihood: -133.122
it: 1  lambda: 0.40  p1: 0.90  p2: 0.10  log-likelihood: -98.270
it: 2  lambda: 0.40  p1: 0.90  p2: 0.10  log-likelihood: -98.270
EM has converged
```




```
In [29]: import time

         startTime = time.process_time()
         EM(utils.generateData(0.8,0.4,0.6,10000,20),0.2,0.7,0.6,False)
         endTime = time.process_time()

         print("Processing time is: ",endTime-startTime," s")
```

```
it: 0  lambda: 0.20  p1: 0.70  p2: 0.60  log-likelihood: -14.909
it: 1  lambda: 0.07  p1: 0.59  p2: 0.43  log-likelihood: -13.686
it: 2  lambda: 0.08  p1: 0.63  p2: 0.42  log-likelihood: -13.672
it: 3  lambda: 0.09  p1: 0.65  p2: 0.42  log-likelihood: -13.668
it: 4  lambda: 0.10  p1: 0.65  p2: 0.42  log-likelihood: -13.667
it: 5  lambda: 0.11  p1: 0.65  p2: 0.41  log-likelihood: -13.666
EM has converged
Processing time is:  0.14015542599999975  s
```



We could not find any parameter set where our EM algorithm takes a particular long time to converge, under the given specified criterion. With increasing N the time does obviiusly increase, due the increase number of samples

In [ ]: