Lab5_ParameterEstimation_TotallySolved

July 6, 2020

```
In [3]: import numpy as np
        import scipy.stats
        import matplotlib.pyplot as plt
```

1 Parameter Estimation: Maximum Likelihood Estimate (MLE)

Given a sample of i.i.d. observations $X = (x_1, x_2, x3, \ldots, x_n)$ from a domain $\mathcal{X}$, where each observation is drawn independently from the domain with the same probability distribution.

Density estimation involves selecting a model, i.e. a probability distribution function, and search for the parameters of that distribution that best explain the joint probability distribution of the observed data X.

Questions: - How do you choose the probability distribution function? - How do you choose the parameters for the probability distribution function?

Maximum Likelihood Estimation (MLE) involves treating the problem as an **optimization** (or search) **problem**, where we seek a set of parameters that results in the best fit for the joint probability of the data sample X.

- Likelihood: $\mathcal{L}(X \mid \theta) := Pr(X \mid \theta)$ where θ are the parameters of a chosen probability density function

$$\mathcal{L}(X \mid \theta) = \prod_{i=1}^{n} Pr(x_i \mid \theta)$$

- Log-likelihood: Multiplying many small probabilities together can be numerically unstable in practice, therefore, it is common to restate this problem as the sum of the log conditional probabilities of observing each example given the model parameters

$$\log \mathcal{L}(X \mid \theta) = \sum_{i=1}^{n} \log(Pr(x_i \mid \theta))$$

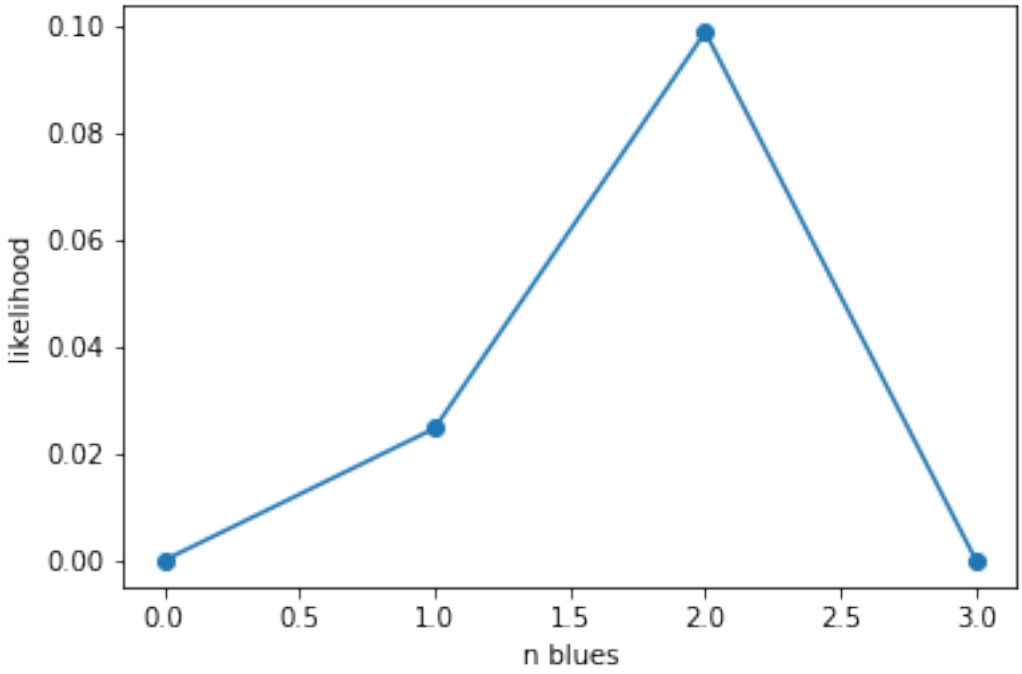
1.0.1 1.1 Discrete distribution: Bernoulli

I have a bag that contains 3 balls. Each ball is either red or blue, but I have no information in addition to this. Thus, the number of blue balls, call it , might be 0, 1, 2, or 3. I am allowed to choose 4 balls at random from the bag with replacement. We define the random variables X1, X2, X3, and X4 as follows

Urn containing N balls. Each ball is either blue or red. Let *n_blue* denotes the number of blue balls in the urn (unknown). We are allowed to observe n balls from the urn (with replacement). We define the random variables x_i for $i = 1, \ldots, n$ such that $x_i = 1$ if the i-th chosen ball is blue and $x_i = 0$ if it is red.

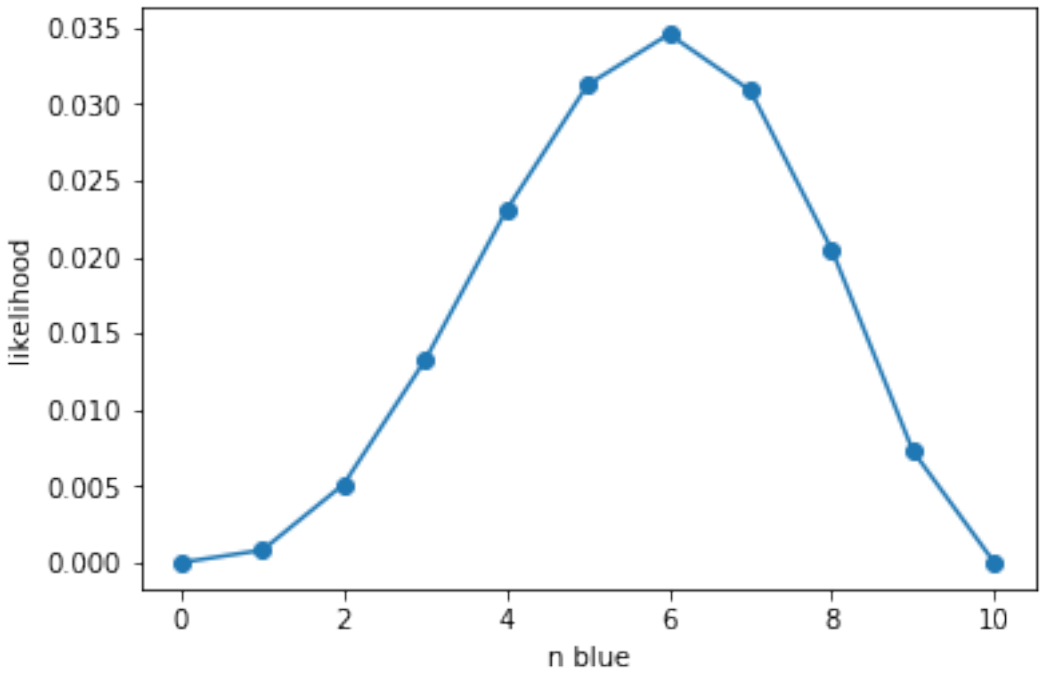
For instance, assume that the urn contains 3 balls and that we observe the following values: $x1 = 1, x2 = 0, x3 = 1, x4 = 1$. 1. For each possible value of *n_blues*, find the probability of the observed sample, $(x1, x2, x3, x4) = (1, 0, 1, 1)$. 2. For which value of *n_blues* is the probability of the observed sample is the largest?

```
In [4]: N = 3 # number of balls
        obs_data_1 = np.array([1,0,1,1])

In [5]: def lkh_bern(data, N, p):
            # N = number of balls
            lkh = 1
            for d in data:
                if d==1:
                    lkh *= p/N
                else:
                    lkh *= 1-p/N
            return lkh

In [6]: def loglkh_bern(data, N, p):
            # N = number of balls
            loglkh = 0
            for d in data:
                if d==1:
                    loglkh += np.log(p/N)
                else:
                    loglkh += np.log(1-p/N)
            return loglkh

In [7]: p_values = np.arange(N+1)
        lkh_vect = np.array([lkh_bern(obs_data_1,N, pp) for pp in p_values])

In [8]: plt.plot(p_values, lkh_vect,'o-')
        plt.xlabel("n blues")
        plt.ylabel("likelihood")
        plt.show()
```




```
In [9]: N_2 = 10 #number of balls
        n_2 = 5 #number of samples

        n_blue = 7 # unknown
```

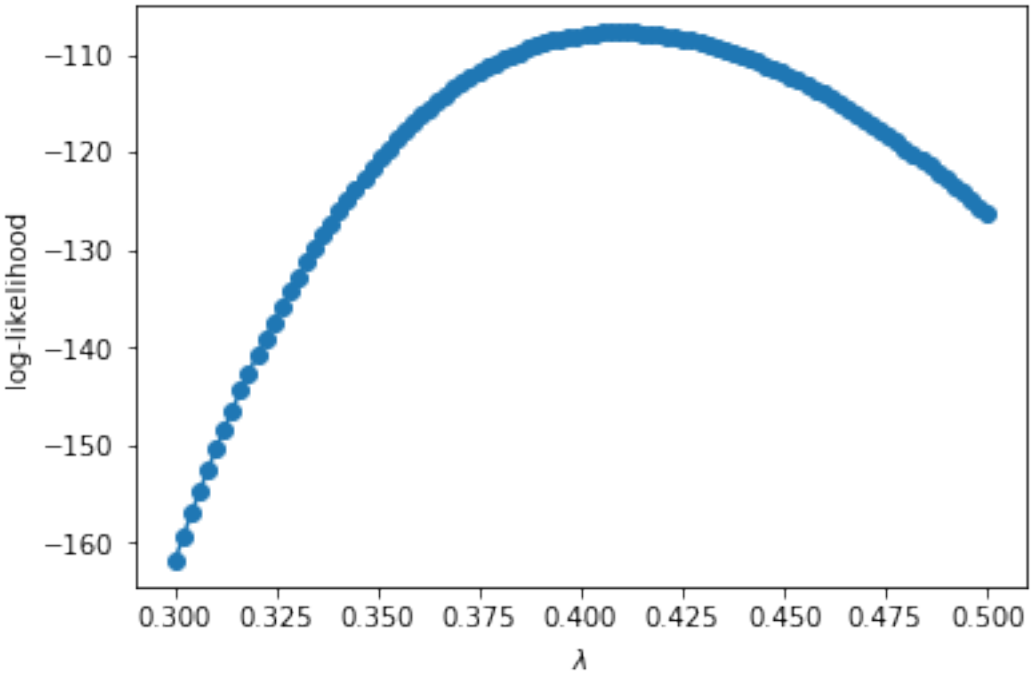
```
In [12]: obs_data_2 = np.random.binomial(1, n_blue/N_2, (n_2,))
         p_values_2 = np.arange(N_2+1)
         lkh_vect_2 = np.array([lkh_bern(obs_data_2,N_2, pp) for pp in p_values_2])
         plt.plot(p_values_2, lkh_vect_2,'o-')
         plt.xlabel("n blue")
         plt.ylabel("likelihood")
         plt.show()
```



1.0.2 2. Continuous distribution: Exponential

Let $X \sim Exp(\lambda)$ be an exponetial distribution and $x_1, \ldots, x_n$ be samples from X. Find the value of *lambda* that better explains the observed data.

```
In [13]: from scipy.stats import expon

In [14]: exp_par = 0.4
         nn = 1000
         data_exp = expon.rvs(scale=exp_par, size=(nn,))

In [15]: def loglkh_exp(data, lamb):
             loglkh = 0
             for d in data:
                 loglkh += np.log(expon.pdf(d, scale = lamb))
             return loglkh

In [16]: n_test = 100
         l_values = np.linspace(0.3,0.5, n_test)
         loglkh_vect = np.array([loglkh_exp(data_exp, l) for l in l_values])

In [17]: plt.plot(l_values, loglkh_vect,'o-')
         plt.xlabel("$\lambda$")
         plt.ylabel("log-likelihood")
         plt.show()
```




```
In [18]: from scipy.optimize import minimize
         # Optimization algorithm

         l0 = 0.3
         obj_exp = lambda l: -loglkh_exp(data_exp,l)
         res = minimize(obj_exp, l0)
         print("REAL: ", exp_par)
         print("ESTIMATED: ", res.x)
```

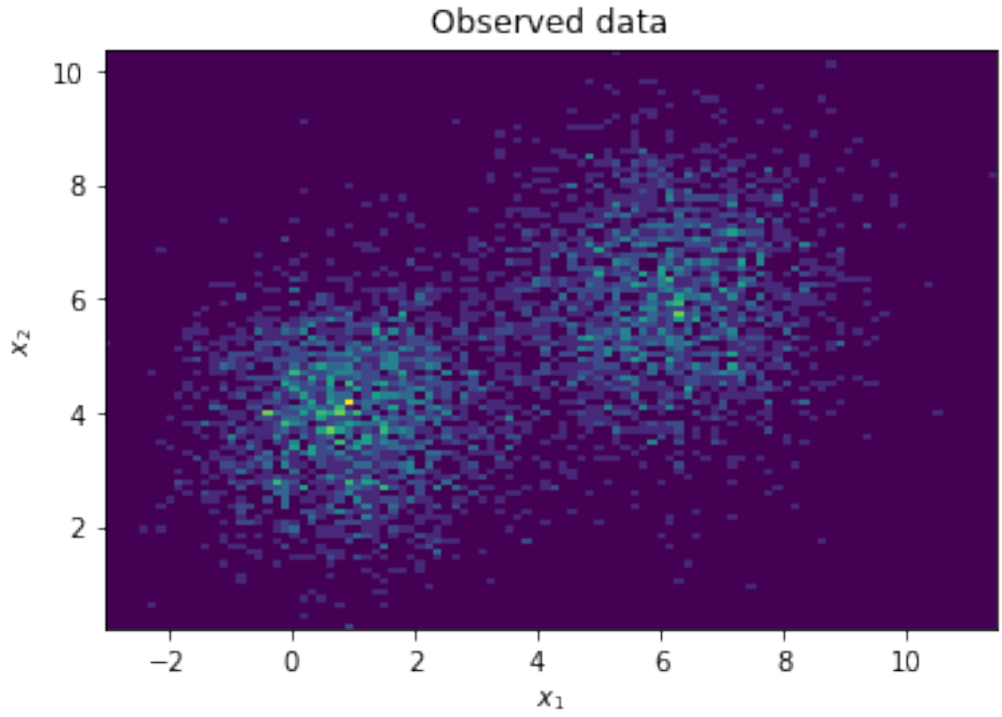
```
REAL:  0.4
ESTIMATED:  [0.40972315]
```

1.0.3 3. General case

Load the data, observe their empirical distribution, choose a model that suits such data and search for the parameters better describe the given data according to the chosen model.

```
In [19]: import pickle

         with open('obs_data.pickle', 'rb') as handle:
             obs_dict = pickle.load(handle)

         observations = obs_dict["observations"]
```

In [20]:
```
plt.hist2d(observations[:,0], observations[:,1], bins=100)
plt.xlabel("$x_1$")
plt.ylabel("$x_2$")
plt.title("Observed data")
plt.show()
```



In [21]:
```
# solution
def log_lkh(theta, data):
    dim = 2

    mu_1 = theta[0:2]
    mu_2 = theta[2:4]
    cov_matrix_1 = np.diag(theta[4]*np.ones(dim))
    cov_matrix_2 = np.diag(theta[5]*np.ones(dim))
    w = theta[6]

    comp_1 = scipy.stats.multivariate_normal(mu_1,cov_matrix_1).pdf(data)
    comp_2 = scipy.stats.multivariate_normal(mu_2,cov_matrix_2).pdf(data)

    return np.sum(np.log((w*comp_1+(1-w)*comp_2)))
```

In [22]:
```
obj_fun = lambda param: -log_lkh(param, observations)
```

In [40]:
```
lb = 1e-10
ub = None
```

```
        AA = (lb, ub)
        BB = (0,1)
        bnds = (AA,AA,AA,AA,AA,AA,BB)
```

In [47]:
```
n_param = 7
theta_0 = np.ones(n_param)
res = minimize(obj_fun, theta_0, bounds=bnds, tol =1e-10, options={'maxiter':1e20, 'di

print("ESTIM", res.x)
```

```
ESTIM [6.05499878 6.02914546 1.00886111 4.00855952 1.94587318 1.51205726
 0.5000307 ]
```

In [ ]: