Homework 2 Solution

AI 539 - Machine Learning for Non-AI Majors
Instructor: Alireza Aghasi

April 25, 2024

Q1. (a) We take n independent samples $x_1, x_2, \ldots, x_n$ from the distribution:

$$f(x) = \frac{2x}{\lambda^2} \exp\left(-\left(\frac{x}{\lambda}\right)^2\right).$$

The goal is estimating the unknown parameterλ.

(a) Derive the maximum likelihood estimates of λ in terms of the samples $x_1, x_2, \ldots, x_n$. Notice that $\lambda \neq 0$, and you would need this assumption to estimate λ without too much work.

(b) Suppose that $n = 3$, and $x_1 = 1.35, x_2 = 2.76, x_3 = 3.12$. Evaluate your ML estimate $\hat{\lambda}$.

Solution. (a) The likelihood function is

$$f(x_1, \ldots, x_n|\lambda) = \frac{2^n \prod_{i=1}^n x_i}{\lambda^{2n}} \exp\left(-\sum_{i=1}^n \left(\frac{x_i}{\lambda}\right)^2\right).$$

By taking the log of the likelihood function we get

$$\begin{aligned} L(\lambda) &= \log\left(f(x_1, \ldots, x_n|\lambda)\right) \\ &= n \log 2 + \sum_{i=1}^n \log x_i - 2n \log \lambda - \sum_{i=1}^n \left(\frac{x_i}{\lambda}\right)^2. \end{aligned}$$

After taking a derivative we get

$$\frac{\partial L}{\partial \lambda} = -\frac{2n}{\lambda} + \frac{2}{\lambda^3} \sum_{i=1}^n x_i^2 = 0.$$

Since $\lambda \neq 0$, we can eliminate a λ from the equation and get

$$n = \frac{1}{\lambda^2} \sum_{i=1}^n x_i^2,$$

which gives

$$\hat{\lambda} = \sqrt{\frac{\sum_{i=1}^n x_i^2}{n}}.$$

(b) By plugging in the values we get

$$\hat{\lambda} = 2.528.$$

Q2. We take n independent samples $x_1, x_2, \dots, x_n$ from the distribution

$$f(x) = \frac{\exp\left(-\frac{x-\alpha}{\beta}\right)}{\beta\left(1+\exp\left(-\frac{x-\alpha}{\beta}\right)\right)^2}.$$

The goal is estimating the parameters α and β based on the observed samples $x_1, x_2, \dots, x_n$. We would not be able to find closed-form expressions for the ML estimates of α and β, and need to consider a numerical minimization technique. Let's go through this task step by step.

(a) Formulate the likelihood function

$$f(x_1, \dots, x_n|\alpha, \beta) = \dots$$

$$f(x_1, \dots, x_n|\alpha, \beta) = \frac{1}{\beta^n}\prod_{i=1}^{n}\frac{\exp\left(-\frac{x_i-\alpha}{\beta}\right)}{\left(1+\exp\left(-\frac{x_i-\alpha}{\beta}\right)\right)^2}.$$

(b) Formulate the negative log-likelihood function

$$L(\alpha, \beta) = -\log\left(f(x_1, \dots, x_n|\alpha, \beta)\right).$$

To estimate the ML estimates $\hat{\alpha}$ and $\hat{\beta}$ we can minimize the negative log likelihood function $L(\lambda, \alpha)$. For this purpose we decide to use a gradient descent scheme. Derive the expression for the gradient components below:

$$\frac{\partial L}{\partial \alpha} = \dots, \qquad \frac{\partial L}{\partial \beta} = \dots. \tag{1}$$

$$L(\alpha, \beta) = -\log\left(f(x_1, \dots, x_n|\alpha, \beta)\right) = n\log\beta + \sum_{i=1}^{n}\frac{x_i-\alpha}{\beta} + 2\sum_{i=1}^{n}\log\left(1+\exp\left(-\frac{x_i-\alpha}{\beta}\right)\right).$$

Taking a derivative with respect to α gives

$$\frac{\partial L}{\partial \alpha} = -\frac{n}{\beta} + 2\sum_{i=1}^{n}\frac{\beta^{-1}\exp\left(-\frac{x_i-\alpha}{\beta}\right)}{1+\exp\left(-\frac{x_i-\alpha}{\beta}\right)},$$

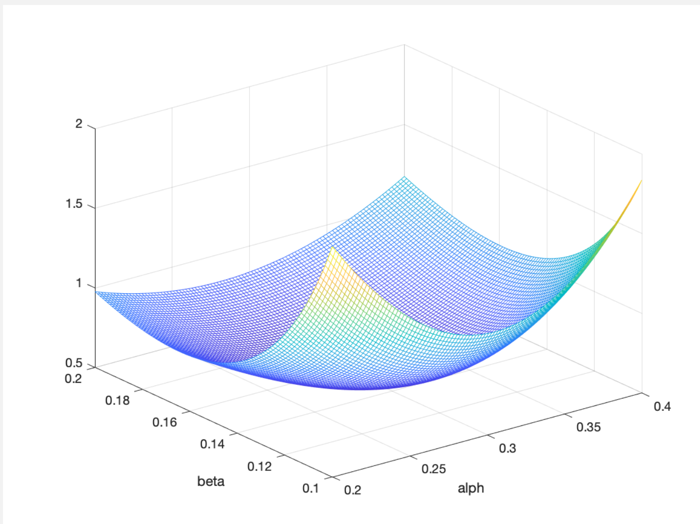
$$\frac{\partial L}{\partial \beta} = \frac{n}{\beta} - \frac{1}{\beta^2}\sum_{i=1}^{n}(x_i-\alpha) + 2\sum_{i=1}^{n}\frac{\frac{x_i-\alpha}{\beta^2}\exp\left(-\frac{x_i-\alpha}{\beta}\right)}{1+\exp\left(-\frac{x_i-\alpha}{\beta}\right)}.$$

(c) Suppose that $n = 5$, and $x_1 = 0.30753, x_2 = 0.56678, x_3 = -0.25177, x_4 = 0.37243, x_5 = 0.26375$. Use the Matlab or Python 3D surface tools to plot $L(\alpha, \beta)$ as a function of α and β, in the following region:

$$0.2 \leq \alpha \leq 0.4, \qquad 0.1 \leq \beta \leq 0.2.$$

Solution. We use MATLAB to do this:

```
x = [.30753, .56678, -.25177, .37243,  .26375];
n = length(x);
[alpha, beta] = meshgrid(linspace(0.2,.4,100),linspace(0.1,.2,100));

sumterm1 = 0;
sumterm2 = 0;
for i = 1:n
sumterm1 = sumterm1 + (x(i) - alpha)./beta;
sumterm2 = sumterm2 + log(1+exp(-(x(i) - alpha)./beta));
end

L = n*log(beta)  + sumterm1 + 2*sumterm2;
mesh(alpha,beta,L)
xlabel('alph')
ylabel('beta')
```

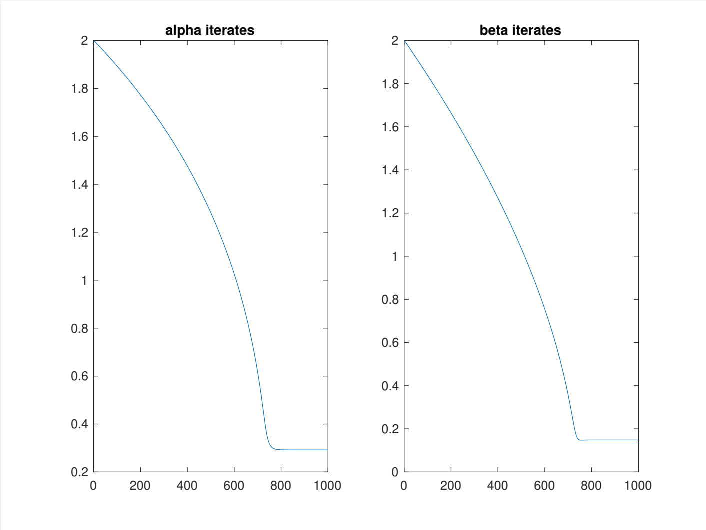


(d) Again, suppose that $n = 5$, and $x_1 = 0.30753, x_2 = 0.56678, x_3 = -0.25177, x_4 = 0.37243, x_5 = 0.26375$. Pick a programming language of your choice and write up a gradient descent (GD) scheme using the gradient components you derived in (1). For your scheme set the gradient learning rate to $\eta = 0.001$, and use the initial values $\alpha_0 = 2$ and $\beta_0 = 2$. Run the GD scheme for 1000 iterates, and report the final estimates of α and β. Also provide a plot showing the iterative values of α for the iterates from 1 to 1000. Provide a similar plot showing the iterative values of β for the iterates from 1 to 1000. Also attach your code.

Solution. The ultimate answer is $\hat{\alpha} = 0.2921$ and $\hat{\beta} = 0.1483$. Below is the MATLAB code:

```
close all
clear all;

x = [.30753, .56678, -.25177, .37243,  .26375];
n = length(x);

grad = @(alpha,beta) [-n/beta+2*sum((exp(-(x-alpha)./beta)/beta)./...
(1+exp(-(x-alpha)./beta))); ...
n/beta-(sum(x-alpha))/(beta^2)+2*sum((exp(-(x-alpha)./beta).* ...
(x-alpha)./(beta^2) )./(1+exp(-(x-alpha)./beta))) ];

% initialization
p = [2;2];
eta = 0.001;

alphIterate = zeros(1000,1);
betIterate = zeros(1000,1);

for i = 1 : 1000
alphIterate(i) = p(1);
betIterate(i) = p(2);
p = p - eta*grad(p(1),p(2));
end

%printing the answer
p
% the result is:
%p =
%  0.2921
%  0.1483

figure;
subplot(121)
plot(1:1000,alphIterate);title('alpha iterates')
subplot(122)
plot(1:1000,betIterate);title('beta iterates')
```

Below is the MATLAB output plot:



(e) Use an exact same setup as part (d), but this time in your gradient descent scheme use a momentum term of $\gamma = 0.9$. Report the final estimates of α and β. Also provide a plot showing the iterative values of α for the iterates from 1 to 1000. Provide a similar plot showing the iterative values of β for the iterates from 1 to 1000. Comparing these plots with those in part (d), which scheme seems to have converged faster? Attach your code.

Solution.

```
close all
clear all;

x = [.30753, .56678, -.25177, .37243,  .26375];
n = length(x);

grad = @(alpha,beta) [-n/beta+2*sum((exp(-(x-alpha)./beta)/beta)./...
(1+exp(-(x-alpha)./beta))); ...
n/beta-(sum(x-alpha))/(beta^2)+2*sum((exp(-(x-alpha)./beta).* ...
(x-alpha)./(beta^2) )./(1+exp(-(x-alpha)./beta))) ];

% initialization
p = [2;2];
eta = 0.001;
gamma = 0.9;
theta = [0;0];

alphIterate = zeros(1000,1);
betIterate = zeros(1000,1);

for i = 1 : 1000
alphIterate(i) = p(1);
betIterate(i) = p(2);
theta = gamma*theta + eta*grad(p(1),p(2));
p = p - theta;
end

%printing the answer
p
% the result is:
%p =
%  0.2921
%  0.1483

figure;
subplot(121)
plot(1:1000,alphIterate);title('alpha iterates')
subplot(122)
plot(1:1000,betIterate);title('beta iterates')
```

Below is the MATLAB output plot. Based on the comparison of the plots with part (d), we clearly see that the scheme with momentum is faster.