STAT 391 Homework 4

Chongyi Xu
University of Washington
STAT 391 Spring 2018
chongyix@uw.edu

---

1. Problem 1 - Rescuing Rob

a Estimate the parameters μ and σ^2 of the normal density that best fits the data in the file hw4-ugrad.dat by the Maximum Likelihood method.

The Maximum Likelihood estimation basiclly calculate for the mean and the variance of the given data $\mathbb{X}$ and use the sample mean as μ and the sample variance as σ^2

```
from statistics import *

# Read in files
# a) hw4-ugrad.dat
dir =
    r'C:\Users\johnn\Documents\UW\SchoolWorks\2018Spring\STAT391\HW4'
f = open(dir+r'\hw4-ugrad.dat')

x = [float(xx) for xx in f.readline().split(' ')]

# ML mu is the mean of data
mu = mean(x)

# ML sigma is the variance of the data
sigma = stdev(x)
```

```
print('The estimated mu is ', mu)
print('And the estimated sigma^2 is ', sigma**2)
```

And the result I got is

```
The estimated mu is  10.018379536
And the estimated sigma^2 is  1.029920746065191
```

b Estimate the parameters a, b of the logistic density that best fits the data in the file hw4-boiler.dat by the Maximum Likelihood method. Make a plot of the log-likelihood of the data at each iteration.

From lecture notes (6.24), we obtained the log-likelihood of the logistic density distribution,

$$l(a,b) = nln(a) - a\sum_i x_i - nb - 2\sum_i ln(1+e^{-ax_i-b})$$

And therefore, the partial derivatives with respecting to a, b are

$$\frac{\partial l}{\partial a} = \frac{n}{a} - \sum_i x_i \frac{1-e^{-ax_i-b}}{1+e^{-ax_i-b}}$$
$$\frac{\partial l}{\partial b} = -\sum_i \frac{1-e^{-ax_i-b}}{1+e^{-ax_i-b}}$$

Since the system is not able to solved analytically, we have to guess for the solution by gradient ascent. The estimated $\hat{a}, \hat{b}$ are

$$\hat{a} \leftarrow \tilde{a} + \eta \frac{\partial l}{\partial a}$$
$$\hat{b} \leftarrow \tilde{b} + \eta \frac{\partial l}{\partial b}$$

where $\tilde{a} = 1$ and $\tilde{b} = 0$ are the intial guess for a, b.

```
# b) hw4-boiler.dat
f = open(dir+r'\hw4-boiler.dat')
```

```
xb = [float(xx) for xx in f.readline().split(' ')]

step = 0.000001
n = len(xb)
iterations = 50000
ll = [0.0]*(iterations-1)
a = [0.0]*iterations
b = [0.0]*iterations
tol = 0.00001
a[0] = 1

def ll_logistic(a, b, x, n):
    return n*np.log(a) - a*sum(x) - n*b - \
           2*sum(np.log(1 + np.exp([-a*xi-b for xi in x])))
def grad_a(a, b, x, n):
    return n/a - sum([xi*(1-np.exp(-a*xi-b))\
                  /(1+np.exp(-a*xi-b)) for xi in x])
def grad_b(a, b, x):
    return -sum((1-np.exp([-a*xi-b for xi in x]))\
                  /(1+np.exp([-a*xi-b for xi in x])))

for i in range(iterations-1):
    ll[i] = ll_logistic(a[i], b[i], xb, n)
    ga = grad_a(a[i], b[i], xb, n)
    gb = grad_b(a[i], b[i], xb)
    if (abs(ga) < tol and abs(gb) < tol):
        break
    a[i+1] = a[i] + step*ga
    b[i+1] = b[i] + step*gb

index,=np.where(np.array(ll)==max(ll))
ll = ll[0:index[0]+1]
a_ML = a[index[0]]
b_ML = b[index[0]]
print('The index is ', index[0])
print('The estimated a is ', a_ML)
print('The estimated b is ', b_ML)

plt.figure(1)
```

```
plt.plot(np.arange(index[0]+1), ll)
plt.title('Log-likelihood')
plt.xlabel('Iteration')
plt.show()

plt.figure(2)
plt.plot(a,b, 'o', Linewidth=0.8)
plt.title('Parameters a,b')
plt.xlabel('a')
plt.ylabel('b')
plt.show()
```

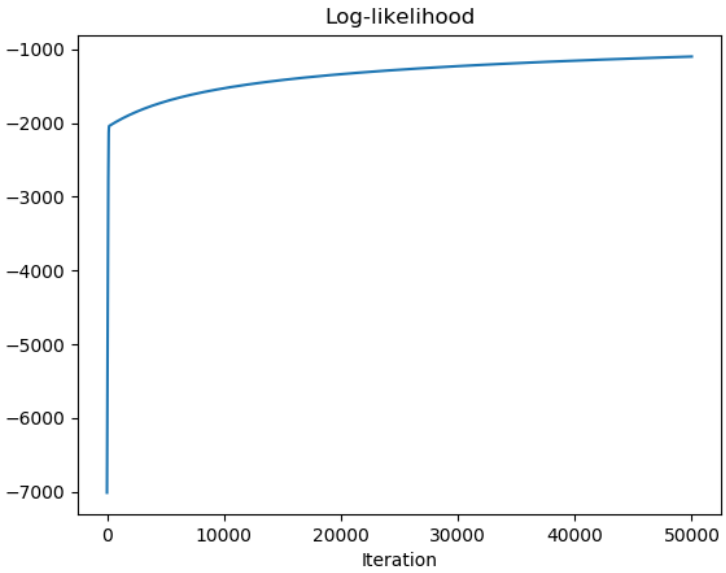
And the plot is Figure 1.

And since we are expecting the Maximum Likelihood estimation of $\hat{a}, \hat{b}$ are $argmaxll(x; a, b)$. And the iteration method will be guaranteed to converge to a local maximum of the log-likelihood, therefore, the a, b are

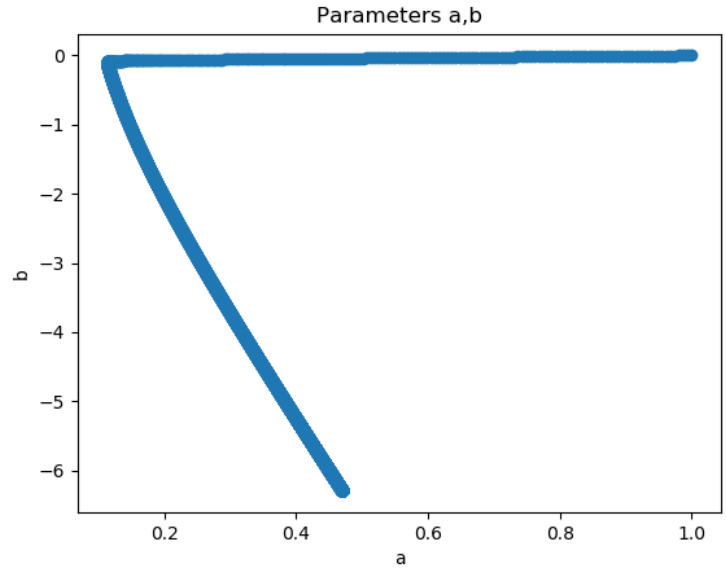
```
i,=np.where(np.array(ll)==max(ll))
a_ML = a[i[0]]
b_ML = b[i[0]]
print('The estimated a is ', a_ML)
print('The estimated b is ', b_ML)
```

```
a_ML = 0.142120071559142
b_ML = -1.992443901848966
```

c Compute a kernel density estimate for the data in the file hw4-coke.dat using a Gaussian kernel with kernel width $h = 0.5$.



(a) The Log-likelihood of the data



(b) The values of paramter a, b

Figure 1: Estimation of logistic density

Since we are using a Gaussian kernel, then

$$
\begin{aligned}
k(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \\
\Rightarrow f_X(x) &= \frac{1}{nh} \sum_{i=1}^{n} k(\frac{x - x_i}{h}) \\
&= \frac{1}{500 \cdot 0.5} \sum_i \frac{1}{\sqrt{2\pi}} exp(\frac{(-\frac{x-x_i}{h})^2}{2}) \\
&= \frac{1}{250\sqrt{2\pi}} \sum_i e^{-2(x-x_i)^2}
\end{aligned}
$$

```
# c). hw4-coke.dat
f = open(dir+r'\hw4-coke.dat')
h = 0.5
xc = [float(xx) for xx in f.readline().split(' ')]
n = len(xc)
```

d After having had his memory restored, Rob suddenly finds himself facing an unknown object whose signature is given in the file hw4-unknown.dat. Help Rob once more: tell him what is the object in front of him.

```
# d). hw4-unknown.dat
f = open(dir+r'\hw4-unknown.dat')

x_test = [float(xx) for xx in f.readline().split(' ')]

classes = ['People', 'Furniture', 'Trash']
score = [0]*len(classes)
score[classes.index('People')] = sum(np.log(norm.pdf(x_test,
    mu, sigma)))
score[classes.index('Furniture')] =
    sum([np.log(a_ML*np.exp(-a_ML*xi-b_ML)/\
                                    (1+np.exp(-a_ML*xi-b_ML)))
                                        for xi in x_test])
for xi in x_test:
```

```
    score[classes.index('Trash')] +=
        np.log(1/(n*h)*sum(1/math.sqrt(2*math.pi)*\
                                    np.exp([-(xi-xci)**2/(2*(h**2))
                                        for xci in xc])))

print('The Log-Likelihood Score for People is ',
    score[classes.index('People')])
print('The Log-Likelihood Score for Furniture is ',
    score[classes.index('Furniture')])
print('The Log-Likelihood Score for Trash is ',
    score[classes.index('Trash')])
print('The Best Score is ', classes[score.index(max(score))],
    ' with ', max(score))
```

```
The Log-Likelihood Score for People is  -485.12788083461237
The Log-Likelihood Score for Furniture is  -791.6883358525606
The Log-Likelihood Score for Trash is  -1018.3103894355676
The Best Score is  Furniture  with  -485.12788083461237s
```

Therefore, the unknown object should be people.

e Make a plot of the densities evaluated in a,b,c and of the data $D_{unknown}$ on the same graph.

The plot is Figure2.

2. Problem 2 - Maximum Likelihood with censored data

a Write the probability that $y_i = 1$ as a function of γ.

$$P(y_i = 1) = P(x_i > 1) = 1 - P(x_i < 1) = 1 - (1 - e^{-\gamma}) = e^{-\gamma}$$

b Derive the expression of the log=likelihood $l(\gamma) = lnP(y_i : n||\gamma)$ as a function of γ

$$\begin{aligned} l(\gamma) &= lnP(y_i : n|\gamma) \\ &= ln[(e^{-\gamma})^{\sum_i y_i}(1 - e^{-\gamma})^{n-\sum_i y_i}] \\ &= ln(e^{-\gamma})\sum_i y_i + ln(1 - e^{-\gamma})(n - \sum_i y_i) \end{aligned}$$

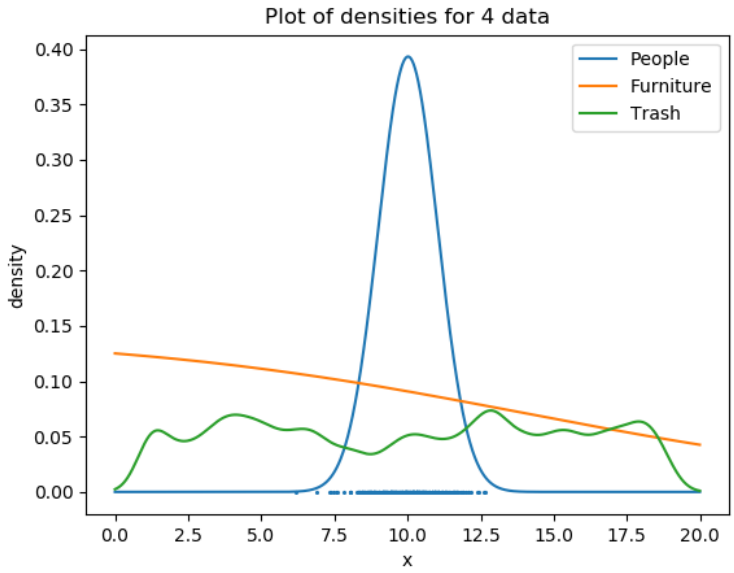


Figure 2: Plot of densities for People, Furniture, Trash. The scatter points are the unknown object, we can see that the scatter points follows the normal distribution pretty well

c Maximize l with respecting to γ and obtain the expression for γ^{ML}. Denote $\theta = e^{-\gamma}$. Then $l(\theta) = ln(\theta)\sum_i y_i + ln(1-\theta)(n - \sum_i y_i)$.

$$\begin{aligned}
\frac{\partial l}{\partial \theta} &= \frac{\sum_i y_i}{\theta} - (n - \sum_i y_i)\frac{1}{1-\theta} = 0 \\
0 &= (1-\theta)\sum y_i - (n - \sum y_i)\theta \\
0 &= \sum y_i - n\theta \\
\theta &= \frac{\sum y_i}{n} \\
\Rightarrow \gamma^{ML} &= -ln(\frac{\sum y_i}{n})
\end{aligned}$$

d Does this problem have sufficient statistics? How many and what are they?

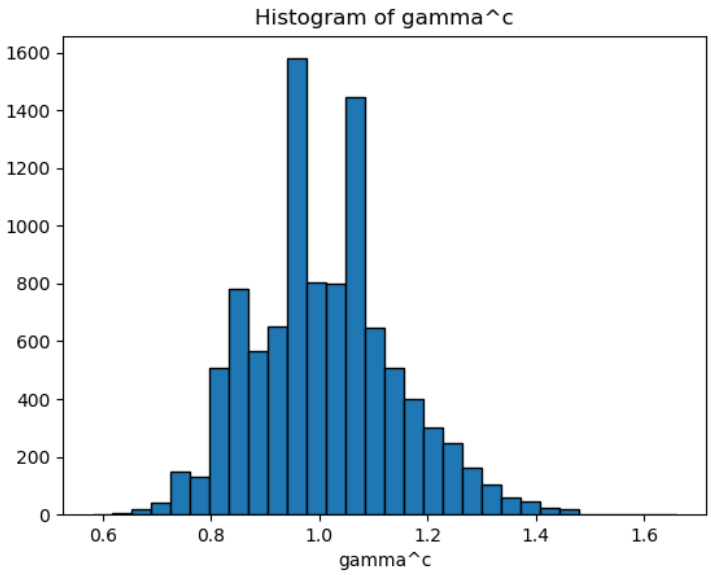
There is only one sufficient statistics in this problem which is the mean of y_i.

e Is the estimate of γ you are obtaining "bettter", "worse", "the same" as you would have obtained had you not lost the original data $x_{1:n}$?

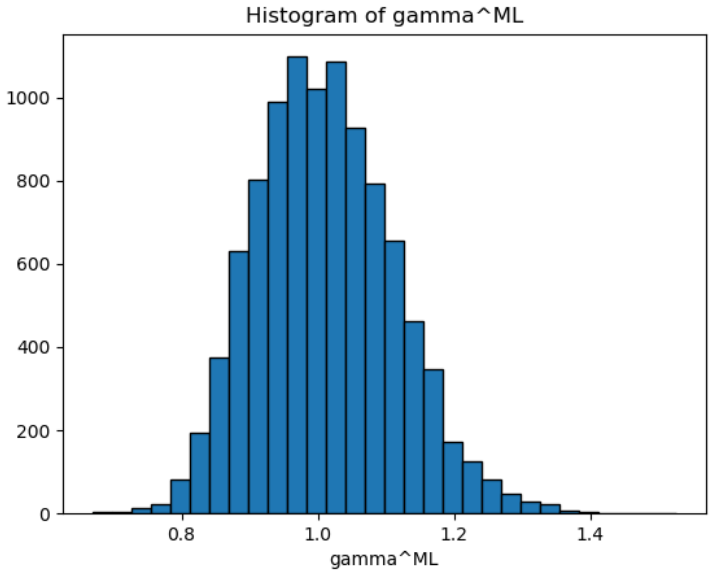
I think that the estimation should be better than using the original data. Since using this method is determing if the output is greater than 1. Comparing to original data, y_i will be less varied which will have less variance. In the other words, it will give a "better" result. The "better" generally means less varied centering at the "real" γ

f Sample $n = 100$ points from an exponential distribution with $\gamma = 1$, censor them, and estimate γ^{ML} from $x_{1:n}$ and γ^c from $y_{1:n}$. Compare the accuracy of the values obtained.

```
import numpy as np
import matplotlib.pyplot as plt
sample_size = 100
np.random.seed(123)
N = 10000
gammaML = [0.0]*N
gammaC = [0.0]*N

for i in range(N):
    x = np.random.exponential(1, sample_size)
    y = (x>1).astype(int)
    gammaML[i] = sample_size/sum(x)
    gammaC[i] = -np.log(sum(y)/sample_size)

plt.figure(1)
plt.hist(gammaC, bins=30, edgecolor='black')
plt.title('Histogram of gamma^c')
plt.xlabel('gamma^c')
plt.show()

plt.figure(2)
plt.hist(gammaML, bins=30, edgecolor='black')
plt.title('Histogram of gamma^ML')
plt.xlabel('gamma^ML')
plt.show()
```



(a) The histogram of γ^c



(b) The histogram of γ^{ML}

Figure 3: From the figure, we can see both centering at $\gamma = 1$, but γ^{ML} is generally less varied rather than γ^c.

The figure is Figure3

3. Estimating h by cross-validation

a Read in the training set D consisting of $n = 1000$ samples from F and validation set D_v of m = 300 samples from

files hw4-f-train.dat, hw4-f-valid.dat. Use a kernel of your choice and then fi

nd the optimal kernel width h by cross-validation. Repeat this for several values of h and plot $L_v(h)$ and $L(h)$ as a function of h on the same graph.
Let h^* be the h that maximizes $L_v(h)$. Make a plot of $f_{h^*}(x)$. Plot the true $f(x)$ on the same graph.

For this question, I will choose the kernel function to be Gaussian function

$$k(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

Therefore, my $f_h(x)$ will be

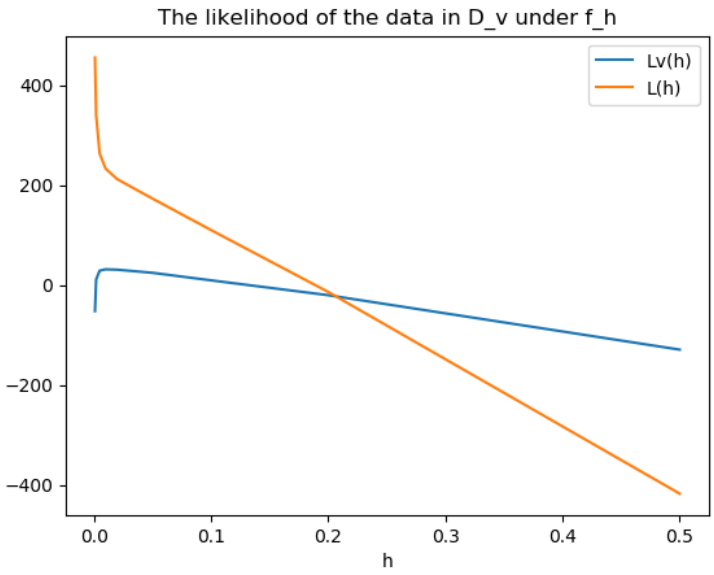
$$f_h(x) = \frac{1}{nh} \sum_i \frac{1}{sqrt2\pi} e^{\frac{-(x-x_i)^2}{2h^2}}$$

```
import numpy as np
import math
import matplotlib.pyplot as plt

def findLL(train, test, h):
    ll = [0.]*len(test)
    n = len(train)
    for i in range(len(test)):
        ll[i] = np.log(1/(n*h)*sum(1/math.sqrt(2*math.pi)*\
                    np.exp([-(test[i] - xi_train)**2/(2*h**2) \
                        for xi_train in train])))
    return ll
```

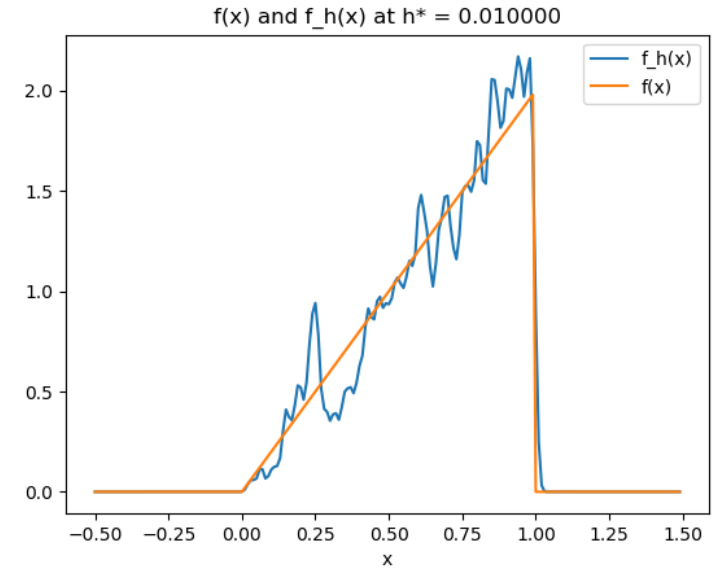
```
dir =
    r'C:\Users\johnn\Documents\UW\SchoolWorks\2018Spring\STAT391\HW4'
f = open(dir+r'\hw4-f-train.dat')
train = [float(xx) for xx in f]

f = open(dir+r'\hw4-f-valid.dat')
valid = [float(xx) for xx in f]

h = [0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5]
Lv = [0.]*len(h)
L = [0.]*len(h)

for k in range(len(h)):
    Lv[k] = sum(findLL(train, valid, h[k]))
    L[k] = sum(findLL(train, train, h[k]))

plt.figure(1)
plt.plot(h, Lv, label='Lv(h)')
plt.plot(h, L, label='L(h)')
plt.title('The likelihood of the data in D_v under f_h')
plt.xlabel('h')
plt.legend()
plt.show()

hmax = h[np.where(Lv==max(Lv))[0][0]]
x = np.arange(-0.5, 1.5, 0.01)
fx = np.array([0.]*len(x))
fx[np.logical_and(x<=1, x>=0)] = 2 * x[np.logical_and(x<=1,
    x>=0)]
plt.figure(2)
plt.plot(x, np.exp(findLL(train, x, hmax)), label='f_h(x)')
plt.plot(x, fx, label='f(x)')
plt.xlabel('x')
plt.title('f(x) and f_h(x) at h* = %f'% hmax)
plt.legend()
plt.show()
```

The Figure4 shows the result. As a result, h^* I found is

```
>>> h[np.where(Lv==max(Lv))[0][0]]
```



(a) The likelihood $L_v(x)$ of the data in D_v and $L(x)$ of the data in training set D



(b) $f_h^*(x)$ with real $f(x)$ at $h^* = 0.01$

Figure 4: The plot shows that the kernel density at h^* is pretty close to the real $f(x)$

0.01

b Same for G and g from the files hw4-g-train.dat, hw4-g-valid.dat.

Basically same code as I did in part (a) with slight changes

```
dir =
    r'C:\Users\johnn\Documents\UW\SchoolWorks\2018Spring\STAT391\HW4'
f = open(dir+r'\hw4-g-train.dat')
train = [float(xx) for xx in f]

f = open(dir+r'\hw4-g-valid.dat')
valid = [float(xx) for xx in f]

h = [0.001, 0.002, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5]
Lv = [0.]*len(h)
L = [0.]*len(h)

for k in range(len(h)):
    Lv[k] = sum(findLL(train, valid, h[k]))
    L[k] = sum(findLL(train, train, h[k]))

plt.figure(1)
plt.plot(h, Lv, label='Lv(h)')
plt.plot(h, L, label='L(h)')
plt.title('The likelihood of the data in D_v under g_h')
plt.xlabel('h')
plt.legend()
plt.show()

hmax = h[np.where(Lv==max(Lv))[0][0]]
x = np.arange(-0.5, 1.5, 0.01)
gx = np.array([0.]*len(x))
for i in range(len(x)):
    if x[i]>=0 and x[i]<=0.5:
        gx[i] = 4*x[i]
    elif x[i]>=0.5 and x[i]<= 1:
        gx[i] = 4*(1-x[i])
```

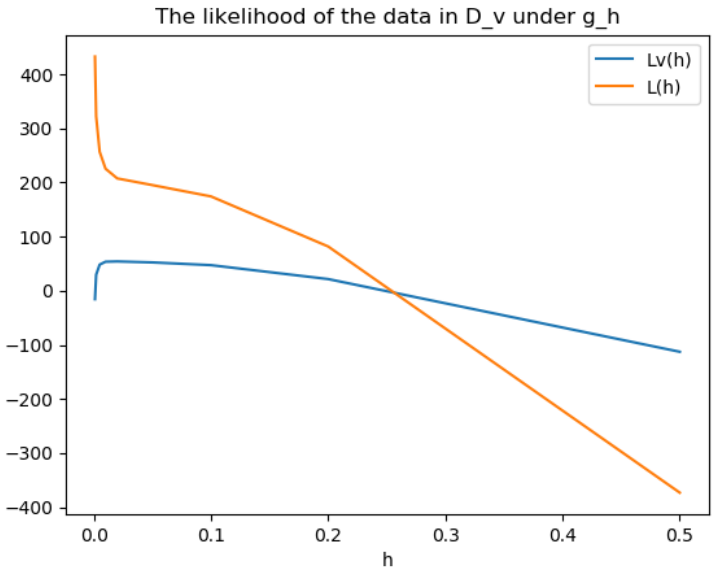
```
plt.figure(2)
plt.plot(x, np.exp(findLL(train, x, hmax)), label='g_h(x)')
plt.plot(x, gx, label='g(x)')
plt.xlabel('x')
plt.title('g(x) and g_h(x) at h* = %f'% hmax)
plt.legend()
plt.show()
```

The plot is Figure5 and the h^* we have for $g(x)$ is

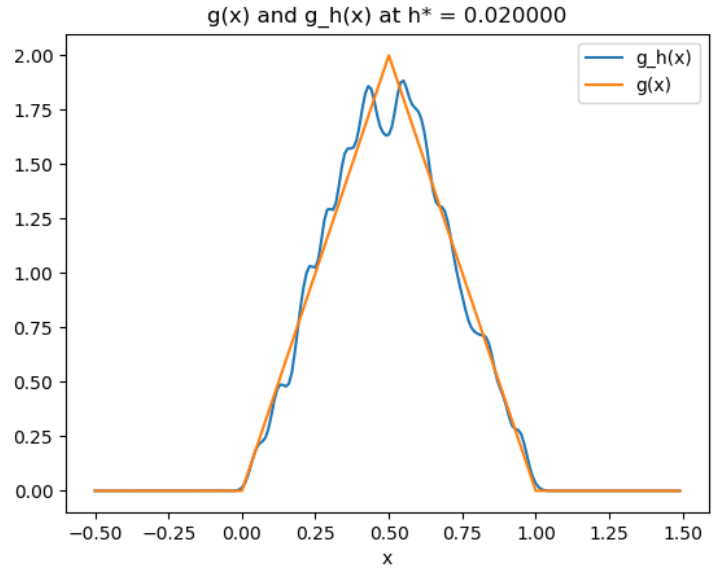
```
>>> h[np.where(Lv==max(Lv))[0][0]]
0.02
```

c Compare the optimal h's and the quality of the plots in a,b. Which of the densities looks easier to approximate? Which of the optimal kernels widths is larger?

From our result, we find out that $g(x)$ is somehow easier to approximate rather than $y(x)$. And the optimal kernel width for $g(x)$ is larger than $y(x)$. The reason might be that for larger width the variance of esimated density will be less.



(a) The likelihood $L_v(x)$ of the data in D_v and $L(x)$ of the data in training set D



(b) $g_h^*(x)$ with real $g(x)$ at $h^* = 0.02$

Figure 5: The plot shows that the kernel density at h^* is pretty close to the real $g(x)$, even better than what we have for $f(x)$.