Machine Learning 1 exercises 2

Jiayun

February 25, 2021

1 Maximum Likelihood Estimation

1.a Show that x and y are independent.

We need to show $p(x, y) = p(x)p(y)$, given $p(x, y) = \lambda\eta e^{-\lambda x - \eta y}$ and $\lambda, \eta > 0$.

$$p(x) = \int_0^{+\infty} p(x, y)dy = -\lambda e^{-\lambda x}[e^{-\eta y}]_0^{+\infty} = \lambda e^{-\lambda x}$$

$$p(y) = \int_0^{+\infty} p(x, y)dx = -\eta e^{-\eta y}[e^{-\lambda x}]_0^{+\infty} = \eta e^{-\eta y}$$

which indicates $p(x)p(y) = \lambda e^{-\lambda x}\eta e^{-\eta y} = p(x, y)$, so x and y are independent.

1.b Derive a maximum likelihood estimator of the parameter λ based on D.

First, we write out the likelihood function regarding λ,

$$p(D|\lambda) = \prod_{k=1}^{N} p((x_k, y_k)|\lambda) = \prod_{k=1}^{N} \lambda\eta e^{-\lambda x_k - \eta y_k}$$

Use both sides natural logarithm:

$$\log p(D|\lambda) = \sum_{k=1}^{N} \log\lambda + \log\eta + (-\lambda x_k - \eta y_k) = N\log\lambda + N\log\eta + \sum_{k=1}^{N}(-\lambda x_k - \eta y_k)$$

Proceed the derivative of λ gives,

$$\nabla_\lambda \log p(D|\lambda) = \frac{N}{\lambda} - \sum_{k=1}^{N} x_k$$

set it to 0, then we get the ML estimator for λ,

$$\hat{\lambda} = \frac{N}{\sum_{k=1}^{N} x_k}$$

1.c Derive a maximum likelihood estimator of the parameter λ based on D under the constraint $\eta = \frac{1}{\lambda}$.

We could easily substitute the constraint into the logarithm likelihood in 1.b, and get the equation below:

$$\log p(D|\lambda) = N\log\lambda + N\log\frac{1}{\lambda} + \sum_{k=1}^{N}(-\lambda x_k - \frac{1}{\lambda}y_k) = \sum_{k=1}^{N}(-\lambda x_k - \frac{1}{\lambda}y_k)$$

then the derivative w.r.t λ is:

$$\nabla_\lambda \log p(D|\lambda) = -\sum_{k=1}^{N} x_k + \sum_{k=1}^{N} \frac{y_k}{\lambda^2}$$

Setting it to 0, rearrange the equation, we will get the ML estimator under the constraint:

$$\hat{\lambda} = \sqrt{\frac{\sum_{k=1}^{N} y_k}{\sum_{k=1}^{N} x_k}}$$

1.d Derive a maximum likelihood estimator of the parameter λ based on D under the constraint $\eta = 1 - \lambda$.

substitute the constraint into the logarithm likelihood in 1.b, and get the equation below:

$$\log p(D|\lambda) = N \log(\lambda - \lambda^2) + \sum_{k=1}^{N} -\lambda x_k - (1-\lambda) y_k$$

set derivative w.r.t λ get:

$$N \frac{1-2\lambda}{\lambda - \lambda^2} + \sum_{k=1}^{N} (-x_k + y_k) = 0$$

We take $\sum_{k=1}^{N}(x_k - y_k)$ as ϵ, then it comes with:

$$\epsilon\lambda^2 - (\epsilon - 2N)\lambda + N = 0$$

$$\hat{\lambda} = \frac{(\epsilon - 2N) \pm \sqrt{(\epsilon - 2N)^2 - 4N\epsilon}}{2\epsilon}$$

2 Maximum Likelihood vs. Bayes

2.a State the likelihood function $p(D|\theta)$, that depends on the parameter θ.

Because of the i.i.d presumption of coin tossing, the likelihood function should be:

$$p(D|\theta) = \prod_{k=1}^{N} p(x_k|\theta) = \theta^5 (1-\theta)^2$$

2.b Compute the maximum likelihood solution $\hat{\theta}$, and evaluate for this parameter the probability that the next two tosses are "head", that is, evaluate $p(x_8 = head, x_9 = head|\hat{\theta})$.

use log likelihood and set it's derivative to 0:

$$\log p(D|\theta) = 5 \log \theta + 2 \log(1-\theta)$$

$$\nabla \log p(D|\theta) = \frac{5}{\theta} - \frac{2}{1-\theta} = 0$$

We could easily get $\hat{\theta} = \frac{5}{7}$. Thus what we need to calculate turns to $p(x_8 = head, x_9 = head|\hat{\theta}) = \hat{\theta}^2 = \frac{25}{49} \approx 0.5102$.

2.c Bayesian view on this problem

First we infer the $\hat{\theta}$ by using bayesian method:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{\int_0^1 p(D|\theta)p(\theta)d\theta} = \frac{\theta^5(1-\theta)^2}{\int_0^1 \theta^5(1-\theta)^2 d\theta} = 168\theta^5(1-\theta)^2$$

So if $0 \leq \theta \leq 1$, $p(\theta|D) = 168\theta^5(1-\theta)^2$ else 0. Use the Bayesian second law introduced in lecture, we will get:

$$p(x_8 = head, x_9 = head|D) = \int_0^1 p(x_8 = head, x_9 = head|\theta)p(\theta|D)d\theta = \int_0^1 \theta^2 168\theta^5(1-\theta)^2 d\theta$$

the result of this integration is $\frac{7}{15} \approx 0.4667$

3 Convergence of Bayes Parameter Estimation

3.a Show the variance of the posterior is contained both by the uncertainty of the data mean and of the prior.

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \Rightarrow \sigma_n^2 = \frac{1}{\frac{1}{\frac{\sigma^2}{n}} + \frac{1}{\sigma_0^2}} \leq \frac{1}{\frac{1}{\sigma_0^2}} = \sigma_0^2$$

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \Rightarrow \sigma_n^2 = \frac{1}{\frac{1}{\frac{\sigma^2}{n}} + \frac{1}{\sigma_0^2}} \leq \frac{1}{\frac{1}{\frac{\sigma^2}{n}}} = \frac{\sigma^2}{n}$$

Combining the two inequalities above, we get the result: $\sigma_n^2 \leq \min(\frac{\sigma^2}{n}, \sigma_0^2)$.

3.b Show the mean of the posterior distribution lies somewhere on the segment between the mean of the prior distribution and the sample mean.

$$\mu_n = (\frac{n}{\sigma^2}\hat{\mu}_n + \frac{\mu_0}{\sigma_0^2})\sigma_n^2 = (\frac{n}{\sigma^2}\hat{\mu}_n + \frac{\mu_0}{\sigma_0^2})\frac{\sigma^2\sigma_0^2}{n\sigma_0^2 + \sigma^2} = \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\mu_0$$

If $\hat{\mu}_n \leq \mu_0$:

$$\mu_n \geq \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\hat{\mu}_n = \hat{\mu}_n$$

$$\mu_n \leq \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\mu_0 + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\mu_0 = \mu_0$$

If $\hat{\mu}_n \geq \mu_0$:

$$\mu_n \leq \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2}\hat{\mu}_n = \hat{\mu}_n$$

$$\mu_n \geq \frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2}\mu_0 + \frac{\sigma}{n\sigma_0^2 + \sigma^2}\mu_0 = \mu_0$$

We combine the 4 inequalities above, then get the conclusion, $\min(\mu_0, \hat{\mu}_n) \leq \mu_n \leq \max(\mu_0, \hat{\mu}_n)$.

sheet02

November 10, 2020

1 Maximum Likelihood Parameter Estimation

In this first exercise, we would like to use the maximum-likelihood method to estimate the best parameter of a data density model $p(x|\theta)$ with respect to some dataset $\mathcal{D} = (x_1, \ldots, x_N)$, and use that approach to build a classifier. Assuming the data is generated independently and identically distributed (iid.), the dataset likelihood is given by

$$p(\mathcal{D}|\theta) = \prod_{k=1}^{N} p(x_k|\theta)$$

and the maximum likelihood solution is then computed as

$$\begin{aligned}\hat{\theta} &= \arg\max_{\theta} \; p(\mathcal{D}|\theta) \\ &= \arg\max_{\theta} \; \log p(\mathcal{D}|\theta)\end{aligned}$$

where the log term can also be expressed as a sum, i.e.

$$\log p(\mathcal{D}|\theta) = \sum_{k=1}^{N} \log p(x_k|\theta).$$

As a first step, we load some useful libraries for numerical computations and plotting.

[47]:
```
import numpy
import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
na = numpy.newaxis
```

We now consider the univariate data density model

$$p(x|\theta) = \frac{1}{\pi} \frac{1}{1 + (x - \theta)^2}$$

also known as the Cauchy distribution with fixed parameter $\gamma = 1$, and with parameter θ unknown. Compared to the Gaussian distribution, the Cauchy distribution is heavy-tailed, and this can be useful to handle the presence of outliers in the data generation process. The probability density function is implemented below.

[48]:
```
def pdf(X,THETA):
    return (1.0 / numpy.pi) * (1.0 / (1+(X-THETA)**2))
```

Note that the function can be called with scalars or with numpy arrays, and if feeding arrays of different shape, numpy broadcasting rules will apply. Our first step will be to implement a function that estimates the optimal parameter $\hat{\theta}$ in the maximum likelihood sense for some dataset $\mathcal{D}$.

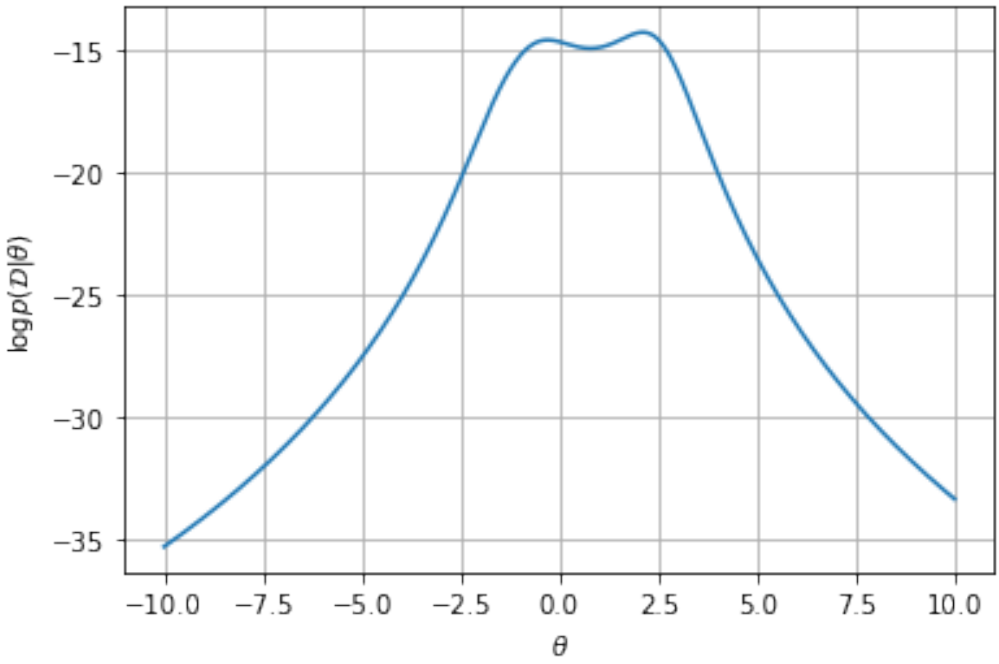
Task (10 P):

- **Implement a function that takes a dataset $\mathcal{D}$ as input (given as one-dimensional array of numbers) and a list of candidate parameters θ (also given as a one-dimensional array), and returns a one-dimensional array containing the log-likelihood w.r.t. the dataset $\mathcal{D}$ for each parameter θ.**

[49]:
```
def ll(D,THETA):

    # ---------------------------------------
    # TODO: replace by your code
    # ---------------------------------------
    LL = 0
    index = len(D)
    for i in range(index):
        LL += numpy.log(pdf(D[i],THETA))
    return LL
    # ---------------------------------------
```

To test the method, we apply it to some dataset, and plot the log-likelihood for some plausible range of parameters θ.

[50]:
```
D = numpy.array([ 2.803, -1.563, -0.853,  2.212, -0.334,  2.503])

THETA = numpy.linspace(-10,10,1001)

plt.grid(True)
plt.plot(THETA,ll(D,THETA))
plt.xlabel(r'$\theta$')
plt.ylabel(r'$\log p(\mathcal{D}|\theta)$')
plt.show()
```



We observe that the likelihood has two peaks: one around $\theta = -0.5$ and one around $\theta = 2$. However, the highest peak is the second one, hence, the second peak is retained as a maximum likelihood solution.

1.0.1 Building a Classifier

We now would like to use the maximum likelihood technique to build a classifier. We consider a labeled dataset where the data associated to the two classes are given by:

[51]:
```
D1 = numpy.array([ 2.803, -1.563, -0.853,  2.212, -0.334,  2.503])
D2 = numpy.array([-4.510, -3.316, -3.050, -3.108, -2.315])
```

To be able to classify new data points, we consider the discriminant function

$$g(x) = \log P(x|\hat{\theta}_1) - \log P(x|\hat{\theta}_2) + \log P(\omega_1) - \log P(\omega_2)$$

were the first two terms can be computed based on our maximum likelihood estimates, and where the last two terms are the prior probabilities. The function $g(x)$ produces the decision ω_1 if $g(x) > 0$ and ω_2 if $g(x) < 0$. We would like to implement a maximum-likelihood based classifier.

Tasks (10 P):

- **Implement the function `fit` that receives as input a vector of candidate parameters θ and the dataset associated to each class, and produces the maximum like-**

lihood parameter estimates. *(Hint: from your function* `fit`*, you can call the function* `ll` *you have previously implemented.)*

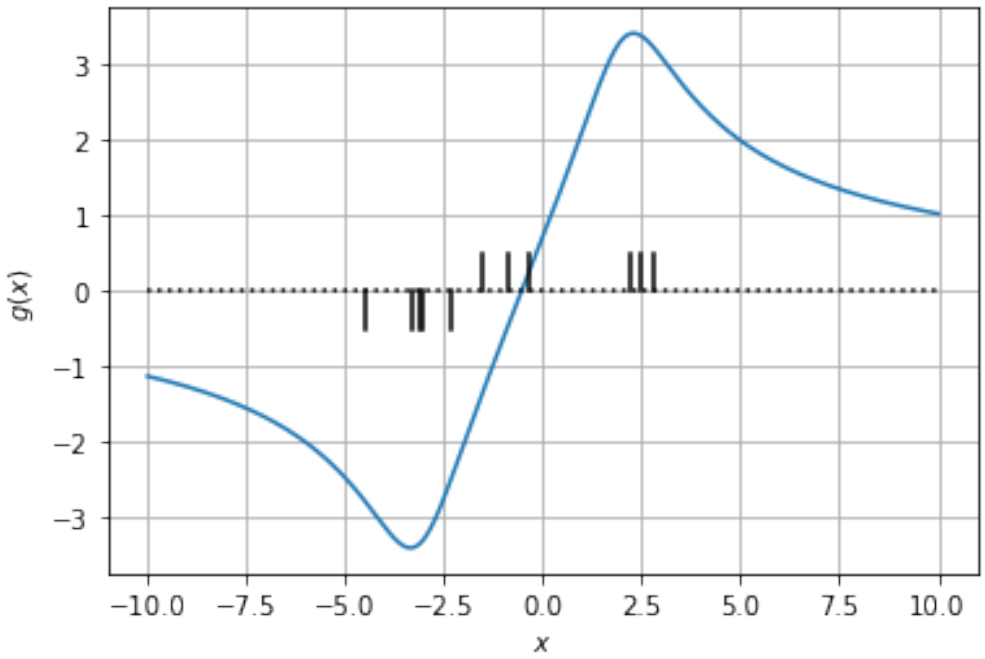
- **Implement the function `predict` that takes as input the prior probability for each class and a vector of points X on which to evaluate the discriminant function, and that outputs a vector containing the value of g for each point in X.**

[52]:
```
class MLClassifier:

    def fit(self,THETA,D1,D2):

        # ---------------------------------------
        # TODO: replace by your code
        # ---------------------------------------
        theta1 = ll(D1, THETA)
        theta2 = ll(D2, THETA)
        rg = numpy.max(THETA) - numpy.min(THETA)
        self.THETA1, self.THETA2 =  numpy.argmax(theta1)/len(THETA)*rg-rg/2, \
                                    numpy.argmax(theta2)/len(THETA)*rg-rg/2
        # ---------------------------------------

    def predict(self,X,p1,p2):

        # ---------------------------------------
        # TODO: replace by your code
        # ---------------------------------------
        g = numpy.log(pdf(X, self.THETA1)) - numpy.log(pdf(X, self.THETA2))\
                                + numpy.log(p1) - numpy.log(p2)
        return g
        # ---------------------------------------
```

Once these two functions have been implemented, the maximum likelihood classifier can be applied to our labeled data, and the decision function it implements can be visualized.

[53]:
```
X = numpy.linspace(-10,10,1001)

plt.grid(True)

mlcl = MLClassifier()
mlcl.fit(THETA,D1,D2)

plt.plot(X,mlcl.predict(X,0.5,0.5))
plt.plot(X,0*X,color='black',ls='dotted')

plt.xlabel(r'$x$')
plt.ylabel(r'$g(x)$')

for d1 in D1: plt.plot([d1,d1],[0,+0.5],color='black')
```

```
for d2 in D2: plt.plot([d2,d2],[0,-0.5],color='black')
```



Here, we observe that the model essentially learns a threshold classifier with threshold approximately -0.5. However, we note that the threshold seems to be too high to properly classify the data. One reason for this is the fact that maximum likelihood estimate retains only the best parameter. Here, the model for the first class focuses mainly on the peak at $x = 2$ and treat examples $x < 0$ as outliers, without considering the possibility that the peak at $\theta = 2$ might actually be the outlier.

2 Bayes Parameter Estimation

Let us now bypass the computation of a maximum likelihood estimate of parameters and adopt instead a full Bayesian approach. We will consider the same data density model and datasets as in the maximum likelihood exercise but we include now a prior distribution over the parameters. Specifically, we set for both classes the prior distribution:

$$p(\theta) = \frac{1}{10\pi} \frac{1}{1 + (\theta/10)^2}$$

Given a dataset $\mathcal{D}$, the posterior distribution for the unknown parameter θ can then be obtained from the Bayes rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta}$$

The integration can be performed numerically using the trapezoidal rule.

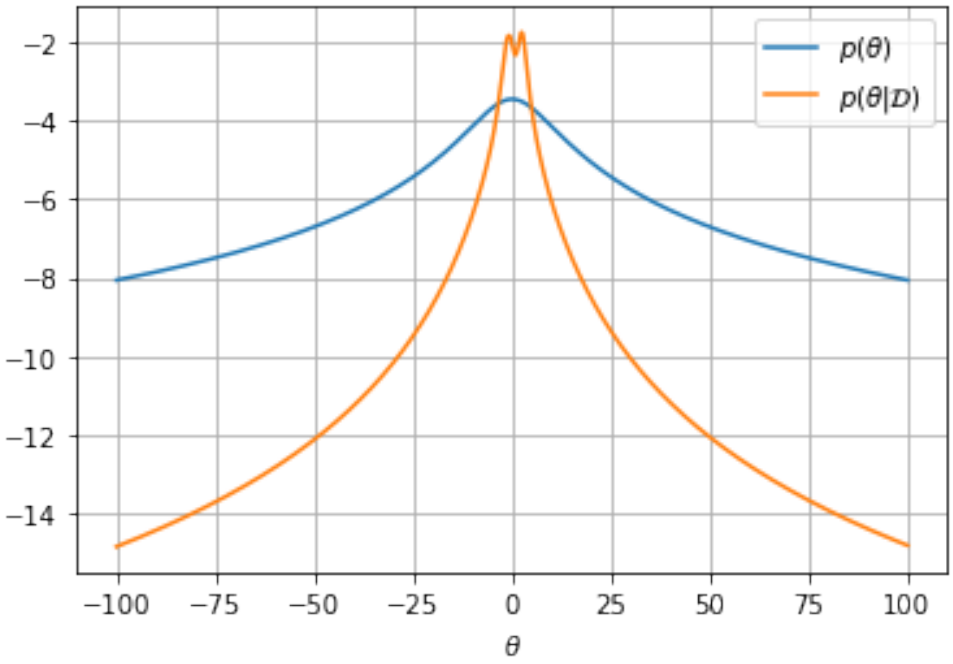
** Task (10 P):**

- **Implement the prior and posterior functions below. These function receive as input a vector of parameters θ (assumed to be sorted from smallest to largest, linearly spaced, and covering the range of values where most of the probability mass lies). The posterior function also receive a dataset $\mathcal{D}$ as input. Both functions return a vector containing the probability scores associated to each value of θ.**

[54]:
```
def prior(THETA):

    # ---------------------------------------
    # TODO: replace by your code
    # ---------------------------------------
    return 1/((10*numpy.pi)*(1+THETA**2/100))
    # ---------------------------------------

def posterior(D,THETA):

    # ---------------------------------------
    # TODO: replace by your code
    # ---------------------------------------
    pri = prior(THETA)
    LL = 0
    index = len(D)
    for i in range(index):
        LL += pdf(D[i],THETA)

    PP = LL*pri
    delta = (numpy.max(THETA)-numpy.min(THETA))/len(THETA)
    de = (PP*delta).sum()

    return PP/de
    # ---------------------------------------
```

To verify the implementation of the two functions, we apply them to the dataset $\mathcal{D}$ defined above and with a broad range of parameters θ.

[55]:
```
THETA = numpy.linspace(-100,100,10001)

plt.grid(True)
plt.plot(THETA,numpy.log(prior(THETA)),label=r'$p(\theta)$')
plt.plot(THETA,numpy.log(posterior(D,THETA)),label=r'$p(\theta|\mathcal{D})$')
plt.legend(); plt.xlabel(r'$\theta$'); plt.show()
```



We observe that the posterior distribution is more concentrated to the specific values of the parameter that explain the dataset well. In particular, we observe the same two peaks around $\theta = -0.5$ and $\theta = 2$ observed in the maximum likelihood exercise.

2.0.1 Building a Classifier

We now would like to build a Bayes classifier based on the discriminant function

$$h(x) = \log P(x|\mathcal{D}_1) - \log P(x|\mathcal{D}_2) + \log P(\omega_1) - \log P(\omega_2)$$

where the dataset-conditioned densities are obtained from the original data density model and the parameter posterior as

$$p(x|\mathcal{D}_j) = \int p(x|\theta)p(\theta|\mathcal{D}_j)d\theta$$

Tasks (10 P):

- **Implement a function `fit` that produces the parameter posteriors $p(\theta|\mathcal{D}_1)$ and $p(\theta|\mathcal{D}_2)$.**
- **Implement a function `predict` computing the new discriminant function h based on the dataset-conditioned data densities.**

[56]:
```
class BayesClassifier:
```

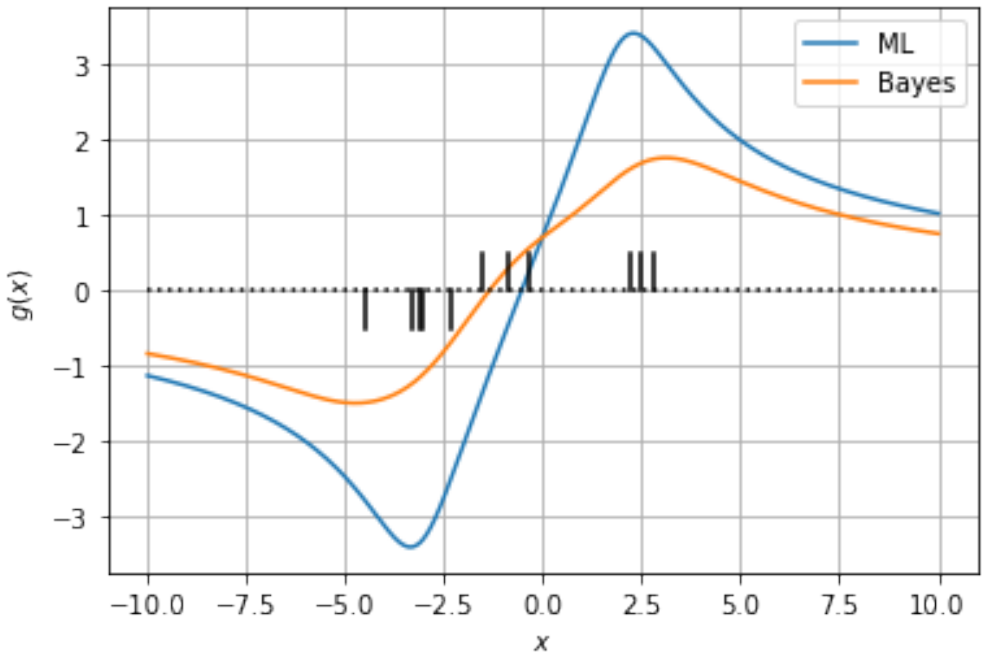
```
    def fit(self,THETA,D1,D2):

        # ---------------------------------------
        # TODO: replace by your code
        # ---------------------------------------
        self.pTHETA_D1 = (posterior(D1,THETA)).reshape(len(THETA),1)
        self.pTHETA_D2 = (posterior(D2,THETA)).reshape(len(THETA),1)
        self.delta = (numpy.max(THETA)-numpy.min(THETA))/len(THETA)
        self.THETA = THETA
        # ---------------------------------------

    def predict(self,X,p1,p2):

        # ---------------------------------------
        # TODO: replace by your code
        # ---------------------------------------
        pX_THETA = numpy.zeros((len(X),len(self.THETA)))

        for index, theta in enumerate(self.THETA):
            temp = pdf(X,theta)
            pX_THETA[:,index] = temp

        pX_D1 = pX_THETA@self.pTHETA_D1  # equal to \sum_{j}␣
↪p(xi|theta_j)*p(theta_j|D1) = p(x|D1)
        pX_D2 = pX_THETA@self.pTHETA_D2
        H = numpy.log(pX_D1) - numpy.log(pX_D2) + p1 - p2
        return H
        # ---------------------------------------
```

We note that the function `predict` is computationally more expensive than the one for maximum likelihood since it involves computing an integral for each point to be predicted.

However, the quality of the prediction also differs compared to that of the maximum likelihood method. In the plot below, we compare the ML and Bayes approaches.

[57]:
```
X = numpy.linspace(-10,10,1001)

bacl = BayesClassifier()
bacl.fit(THETA,D1,D2)

plt.grid(True)
plt.plot(X,mlcl.predict(X,0.5,0.5),label='ML')
plt.plot(X,bacl.predict(X,0.5,0.5),label='Bayes')

plt.plot(X,0*X,color='black',ls='dotted')
plt.xlabel(r'$x$'); plt.ylabel(r'$g(x)$')
plt.legend()
```

```
for d1 in D1: plt.plot([d1,d1],[0,+0.5],color='black')
for d2 in D2: plt.plot([d2,d2],[0,-0.5],color='black')
```



We observe that the Bayes classifier has generally lower output scores and its decision boundary has been noticeably shifted to the left, leading to better predictions for the current data. In this particular case, the difference between the two models can be explained by the fact that the Bayes one better integrates the possibility that negative examples for the first class are not necessarily outliers.