2.

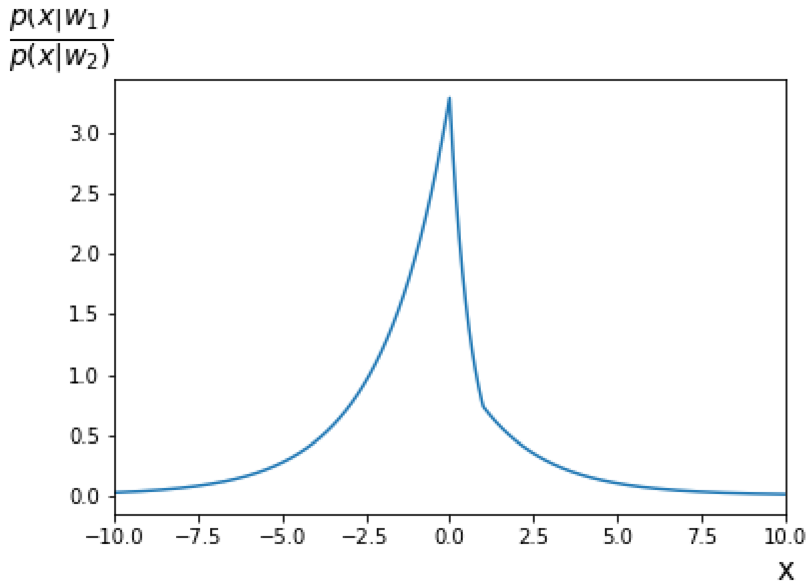
(a)

$$
\begin{aligned}
\int_{-\infty}^{\infty} e^{\frac{-|x-a_i|}{b_i}} dx &= \int_{-\infty}^{a_i} e^{\frac{x-a_i}{b_i}} + \int_{a_i}^{\infty} e^{\frac{-x+a_i}{b_i}} \\
&= b_i e^{\frac{x-a_i}{b_i}} |_{-\infty}^{a_i} - b_i e^{\frac{-x+a_i}{b_i}} |_{a_i}^{+\infty} \\
&= 2b_i
\end{aligned}
$$

(b)

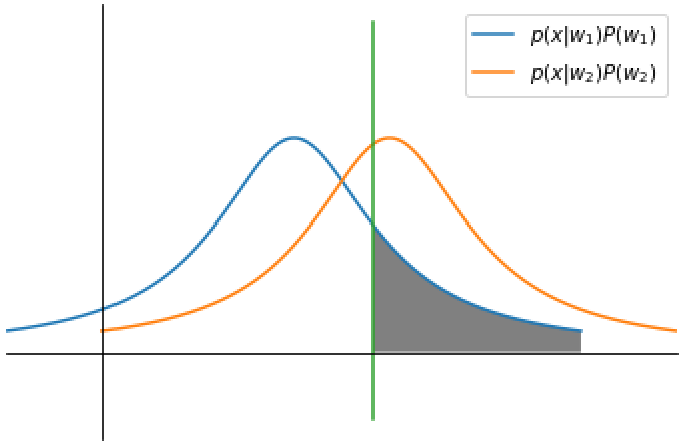
$$
\frac{p(x|w_1)}{p(x|w_2)} = \frac{b_2}{b_1} e^{\frac{|x-a_2|}{b_2} - \frac{|x-a_1|}{b_1}}
$$

(c)



7.

(a)



$$
\begin{aligned}
P(x \in R_2, w_1) &= p(x \in R_2 | w_1)P(w_1) \\
&= \int_{R_2} p(x|w_1)P(w_1)dx \\
&= \frac{1}{2\pi b}\int_B^{\infty} \frac{1}{1 + (\frac{x-a_1}{b})^2} dx \\
&= \frac{1}{2\pi}(\frac{\pi}{2} - arctan\frac{B - a_1}{b}) \\
&= E_1 \\
\therefore B &= \frac{b}{tan(2\pi E_1)} + a_1
\end{aligned}
$$

(b)

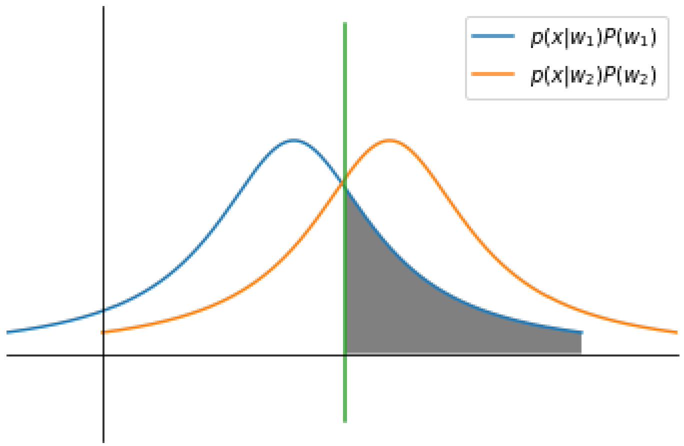
$$
\begin{aligned}
E_2 &= P(x \in R_1, w_2) \\
&= \int_{-\infty}^{B} p(x|w_2)P(w_2)dx \\
&= \frac{1}{2\pi}(\frac{\pi}{2} + arctan(\frac{1}{tan(2\pi E_1)} + \frac{a_1 - a_2}{b}))
\end{aligned}
$$

(c)

$$
E = E_1 + E_2 = E_1 + \frac{1}{2\pi}(\frac{\pi}{2} + arctan(\frac{1}{tan(2\pi E_1)} + \frac{a_1 - a_2}{b}))
$$

(d) Replace b, a_1, a_2 and E_1 with the specified values, we get $E = 0.26125$

(e)



From exercise 2.9(a), we know that Bayes error is the minimum error rate. Neyman-Pearson criterion add some constraints to the origin problem. In order to satisfy these constraints the green vertical line may has to shift and resulting in a higher error rate.

9.

(a)

There is no matter to suppose $a_2 \geq a_1$

$$\begin{aligned} P(error) &= E_1 + E_2 \\ &= p(x \in R_2|w_1)P(w_1) + P(x \in R_1|w_2)P(w_2) \\ &= \frac{1}{2\pi}(\int_{B'}^{\infty} \frac{1}{1+(\frac{x-a_1}{b})^2}dx + \int_{-\infty}^{B'} \frac{1}{1+(\frac{x-a_2}{b})^2}dx) \\ &= \frac{1}{2} + \frac{1}{2\pi}(-arctan\frac{B'-a_1}{b} + arctan\frac{B'-a_2}{b}) \end{aligned}$$

From

$$\frac{dP(error)}{dB'} = 0$$

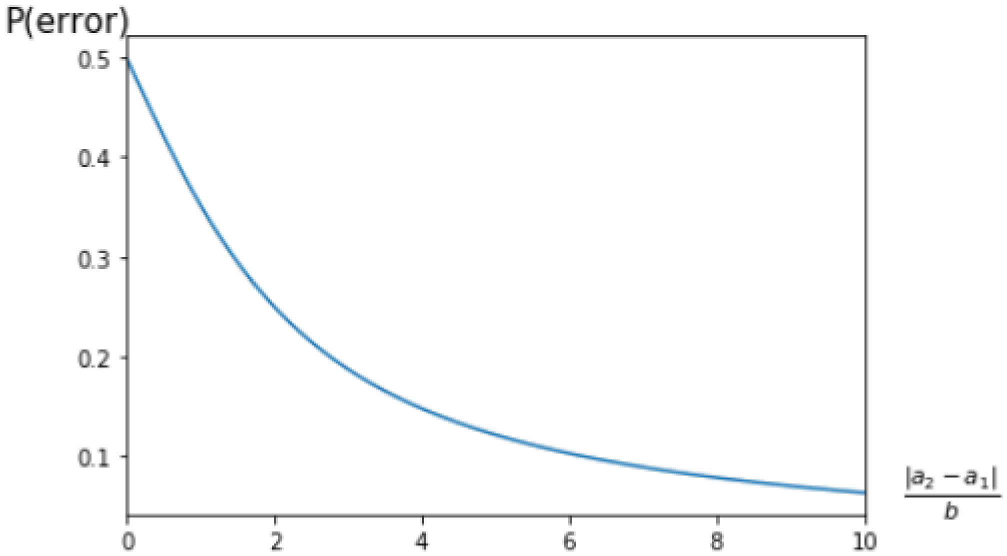
we get

$$minP(error) = \frac{1}{2} - \frac{1}{\pi}arctan(|\frac{a_2-a_1}{2b}|)$$

when

$$B' = \frac{a_1+a_2}{2}$$

(b)



(c)

$maxP(error) = 0.5$ where $a_1 = a_2$. In other words, $p(x|w_1)$ and $p(x|w_2)$ have the same probability distribution. In this situation, the classifier can only give a random guess and has 50% probability to make a mistake. In generative adversarial nets, the generative model is trying to learn the distribution probability of the training data.

12.

(a)

Suppose $P(w_{max}|x) \leq \frac{1}{c}$.

$$\because P(w_{max}|x) \geq P(w_1|x)$$

We have

$$\sum_{i=1}^{c} P(w_i|x) \leq \sum_{i=1}^{c} P(w_{max}|x) < 1$$

contradicts with the condition that $\sum_{i=1}^{c} p(w_i|x) = 1$. Therefore, $P(w_{max}|x) \geq \frac{1}{c}$

(b)

$$
\begin{aligned}
P(error) &= \frac{1}{c}\sum_{j=1}^{c}\sum_{i\neq j} P(x \in R_i, w_j) \\
&= \frac{1}{c}\sum_{j=1}^{c}(1 - \int P(w_{max}|x)p(x)dx) \\
&= 1 - \int P(w_{max}|x)p(x)dx
\end{aligned}
$$

(c)

$$P(error) \leq 1 - \frac{1}{c}\int P(x)dx = \frac{c-1}{c}$$

(d)

$$\forall x, P(w_{max}|x) = \frac{1}{c}$$

23

(a)

$$p(x_0) = \frac{1}{(2\pi)^{\frac{3}{2}}|\Sigma|^{\frac{1}{2}}} e^{-\frac{(x_0-\mu)^2}{2\sigma^2}} = 0.008157$$

(b)

$$\lambda_1 = 7, v_1 = \begin{bmatrix} 0 \\ 0.70710678 \\ 0.70710678 \end{bmatrix}, \lambda_2 = 7, v_2 = \begin{bmatrix} 0 \\ 0.70710678 \\ 0.70710678 \end{bmatrix}, \lambda_3 = 7, v_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\Phi = \begin{bmatrix} 0. & 0. & 1. \\ 0.70710678 & 0.70710678 & 0. \\ 0.70710678 & -0.70710678 & 0. \end{bmatrix}, \quad \Lambda = \begin{bmatrix} 7 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A_w = \Phi\Lambda^{-\frac{1}{2}} = \begin{bmatrix} 0. & 0. & 1. \\ 0.26726124 & 0.40824829 & 0. \\ 0.26726124 & -0.40824829 & 0. \end{bmatrix}$$

(c)

$$p(x_w) = A_w(x_0 - \mu) = \begin{bmatrix} -0.80178373 \\ -0.40824829 \\ -0.5 \end{bmatrix}$$

(d)

$$r_1^2 = (x_0 - \mu)^t \Sigma^{-1}(x_0 - \mu) = 1.0595$$
$$r_2^2 = x_w^t I x_w = 1.0595$$

(e)

$$\mu' = \sum_{k=1}^{n} x_{k'} = \sum_{k=1}^{n} T^t x_k = T^t \sum_{k=1}^{n} x_k = T^t \mu$$

$$\Sigma' = \sum_{k=1}^{n}(x_{k'} - \mu')(x_{k'} - \mu')^t = T^t[\sum_{k=1}^{n}(x_k - \mu)(x_k - \mu)^t]T = T^t\Sigma T$$

After linear transformation, $p(T^t x_0|N(T^t\mu, T^t\Sigma T)) = \frac{p(x_0|N(\mu,\Sigma))}{|T|}$. Therefore, the probability density function is variant under a general linear transformation.

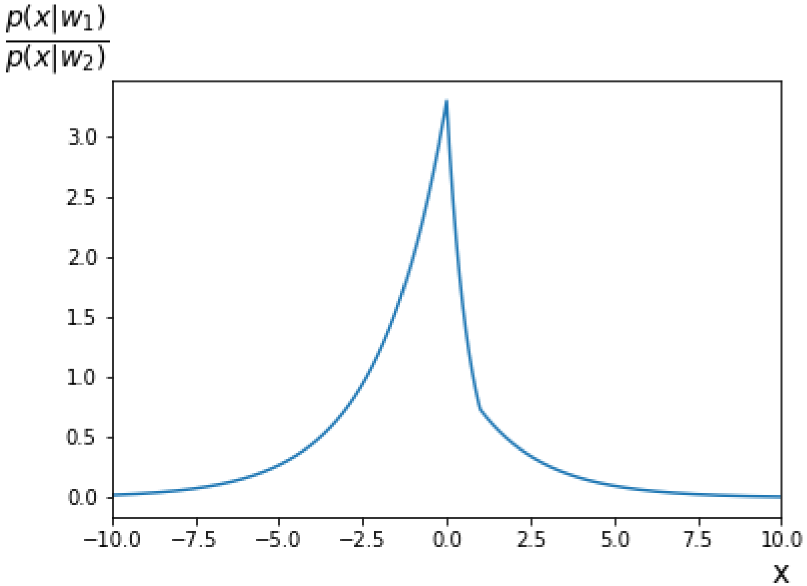
(f)

$$\begin{aligned} Var(x) &= A_w^t \Sigma A_w \\ &= A^{-\frac{t}{2}}\Phi^t\Sigma\Phi A^{-\frac{1}{2}} \\ &= A^{-\frac{t}{2}}\Phi^t\Phi\Lambda\Phi^t\Phi A^{-\frac{1}{2}} \\ &= I \end{aligned}$$

source code

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import math
```

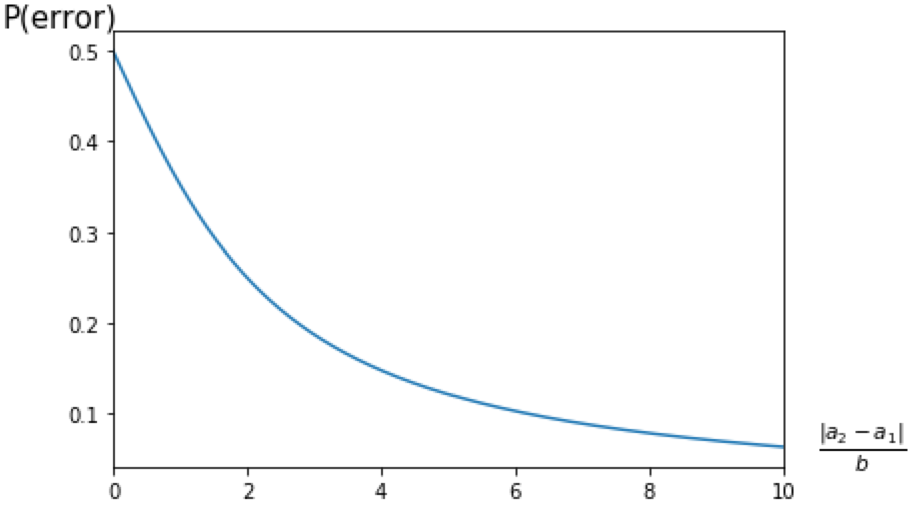
In [2]:
```
x = np.arange(-10.0, 10.0, 0.002)
y = 2 * np.exp(abs(x-1)/2 - abs(x))

plt.xlim(-10, 10)
plt.xlabel('x', position = (1,0), fontsize = 15)
plt.ylabel('$\\frac{p(x|w_1)}{p(x|w_2)}$', rotation = 0,fontsize = 20, position = (0,1))
plt.plot(x,y)
plt.savefig('2c')
plt.show()
```



In [3]:
```
b, a1, a2, E1, pi = 1, -1, 1, 0.1, math.pi
E1 + 1 / (2 * pi) * (pi / 2 + math.atan(1/(math.tan(2*pi*E1)) + (a1-a2)/b))
```

Out[3]: 0.26125441471751243

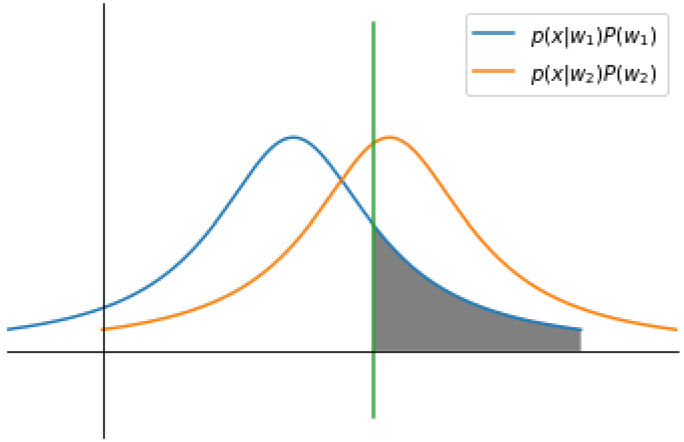
In [4]:
```
x = np.arange(0, 10.0, 0.002)
y = 1 / 2 - 1 / math.pi * np.arctan(x/2)
plt.xlim(0, 10)
#plt.xlabel('$\\frac{|a_2 - a_1|}{b}$', position = (1,0), fontsize = 15)
plt.text(10.5, 0.05, r'$\frac{|a_2 - a_1|}{b}$', fontsize = 15)
plt.ylabel('P(error)', position = (0,1), fontsize = 15, rotation = 0)
plt.plot(x,y)
#plt.text (2, 1, r'$\mu=100$', fontsize = 15)
plt.savefig('9b')
```



In [5]:

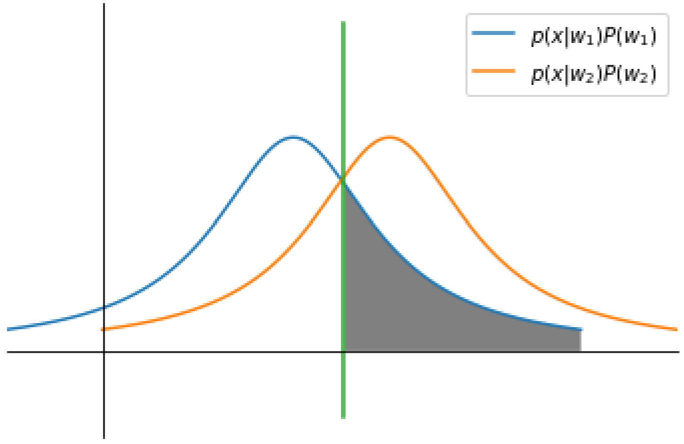
```python
point_number = 1000
axes = plt.subplot(111)
b = 1
width = 3

#The first normal distribution function
a1 = 2
x1 = np.linspace(a1 - width, a1 + width,point_number)
y1 = 1 / (np.pi * b * (1 + ((x1 - a1)/b)**2))
axes.plot(x1,y1, label='$p(x|w_1)P(w_1)$')

#The second normal distribution function
a2 = 3
x2 = np.linspace(a2 - width, a2 + width,point_number)
y2 = 1 / (np.pi * b * (1 + ((x2 - a2)/b)**2))
axes.plot(x2, y2, label='$p(x|w_2)P(w_2)$')

#Find the intersection of these two normal distribution function
solution = []
for i in range(point_number):
    for j in range(point_number):
        if(abs(y1[i] - y2[j]) < 1e-2 and abs(x1[i] - x2[j]) < 1e-2):
            solution = [x1[i], y1[i]]
            break

# Draw vertical line
y3 = np.arange(-0.1, 0.5, 0.01)
x3 = np.zeros((len(y3),1)) + solution[0] + 0.3
axes.plot(x3, y3)

#plt.scatter(1,1)
#plt.axvline(x = 1, ymin = -1, ymax = 0.8)

#fill the picture
axes.fill_between(x1, y1, where=(x3[0]<x1), facecolor='gray')

axes.set_xlim(a1 - width, a2 + width)
#hide the tick
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data',0))
axes.spines['left'].set_position(('data',0))
```

```
axes.set_xticks([])
axes.set_yticks([])
#show legend
plt.legend(loc='upper right')
plt.savefig('7a')
```



In [6]:
```
point_number = 1000
axes = plt.subplot(111)
b = 1
width = 3

#The first normal distribution function
a1 = 2
x1 = np.linspace(a1 - width, a1 + width,point_number)
y1 = 1 / (np.pi * b * (1 + ((x1 - a1)/b)**2))
axes.plot(x1,y1, label='$p(x|w_1)P(w_1)$')

#The second normal distribution function
a2 = 3
x2 = np.linspace(a2 - width, a2 + width,point_number)
y2 = 1 / (np.pi * b * (1 + ((x2 - a2)/b)**2))
axes.plot(x2, y2, label='$p(x|w_2)P(w_2)$')

#Find the intersection of these two normal distribution function
solution = []
for i in range(point_number):
    for j in range(point_number):
        if(abs(y1[i] - y2[j]) < 1e-2 and abs(x1[i] - x2[j]) < 1e-2):
            solution = [x1[i], y1[i]]
            break

# Draw vertical line
y3 = np.arange(-0.1, 0.5, 0.01)
x3 = np.zeros((len(y3),1)) + solution[0]
axes.plot(x3, y3)

#plt.scatter(1,1)
#plt.axvline(x = 1, ymin = -1, ymax = 0.8)

#fill the picture
axes.fill_between(x1, y1, where=(x3[0]<x1), facecolor='gray')

axes.set_xlim(a1 - width, a2 + width)
#hide the tick
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data',0))
axes.spines['left'].set_position(('data',0))
```

```
axes.set_xticks([])
axes.set_yticks([])
#show legend
plt.legend(loc='upper right')
plt.savefig('7e')
```



In [7]:
```
mu = np.array([1,2,2])
sigma = np.array([[1,0,0],[0,5,2],[0,2,5]])
x0 = np.array([0.5,0,1])

inverse = np.linalg.inv(sigma)
determinent = np.linalg.det(sigma)

probability = np.exp(- (np.transpose(x0 - mu) @ inverse @ (x0 - mu)) / 2) / ((2 * np.pi) ** (3/2) * np.sqrt(determi
print(probability)
```

```
0.008157327113891189
```

In [8]:
```python
eigenvalue, eigenvector = np.linalg.eig(sigma)

#swap eigenvalue and eigenvector P[:, [0, 2]] = P[:, [2, 0]]
# eigenvalue[[0,2]] = eigenvalue[[2,0]]
# eigenvector[:,[0,2]] = eigenvector[:,[2,0]]

print(eigenvalue, eigenvector)

eigenvalue = np.linalg.inv(np.sqrt(np.diag(eigenvalue)))
print(eigenvalue)
```

```
[7. 3. 1.] [[ 0.          0.          1.        ]
 [ 0.70710678  0.70710678  0.        ]
 [ 0.70710678 -0.70710678  0.        ]]
[[0.37796447 0.         0.        ]
 [0.         0.57735027 0.        ]
 [0.         0.         1.        ]]
```

In [9]:
```python
# remember to transpose
xw = np.transpose(eigenvector @ eigenvalue) @ (x0-mu)
print(eigenvector @ eigenvalue)
print(x0 - mu)
print(xw)
```

```
[[ 0.          0.          1.        ]
 [ 0.26726124  0.40824829  0.        ]
 [ 0.26726124 -0.40824829  0.        ]]
[-0.5 -2.  -1. ]
[-0.80178373 -0.40824829 -0.5       ]
```

In [10]:
```python
xw20 = np.transpose(xw)  @  xw
print(xw20)
x020 = np.transpose((x0 - mu)) @ inverse @ (x0 - mu)
print(x020)
```

```
1.0595238095238093
1.0595238095238095
```