Exercises for the course

Machine Learning 1

Winter semester 2021/22

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 1

Exercise 1: Estimating the Bayes Error $(10 + 10 + 10$ P$)$

The Bayes decision rule for the two classes classification problem results in the Bayes error

$$P(\text{error}) = \int P(\text{error}|\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x},$$

where $P(\text{error}|\boldsymbol{x}) = \min\left[\, P(\omega_1|\boldsymbol{x})\,,\ P(\omega_2|\boldsymbol{x})\,\right]$ is the probability of error for a particular input $\boldsymbol{x}$. Interestingly, while class posteriors $P(\omega_1|\boldsymbol{x})$ and $P(\omega_2|\boldsymbol{x})$ can often be expressed analytically and are integrable, the error function has discontinuities that prevent its analytical integration, and therefore, direct computation of the Bayes error.

(a) *Show* that the full error can be upper-bounded as follows:

$$P(\text{error}) \le \int \frac{2}{\frac{1}{P(\omega_1|\boldsymbol{x})} + \frac{1}{P(\omega_2|\boldsymbol{x})}}\, p(\boldsymbol{x})\, d\boldsymbol{x}.$$

Note that the integrand is now continuous and corresponds to the harmonic mean of class posteriors weighted by $p(\boldsymbol{x})$.

(b) *Show* using this result that for the univariate probability distributions

$$p(x|\omega_1) = \frac{\pi^{-1}}{1 + (x-\mu)^2} \quad \text{and} \quad p(x|\omega_2) = \frac{\pi^{-1}}{1 + (x+\mu)^2},$$

the Bayes error can be upper-bounded by:

$$P(\text{error}) \le \frac{2\, P(\omega_1)P(\omega_2)}{\sqrt{1 + 4\mu^2 P(\omega_1)P(\omega_2)}}$$

(Hint: you can use the identity $\int \frac{1}{ax^2+bx+c}\, dx = \frac{2\pi}{\sqrt{4ac-b^2}}$ for $b^2 < 4ac$.)

(c) *Explain* how you would estimate the error if there was no upper-bounds that are both tight and analytically integrable. Discuss following two cases: (1) the data is low-dimensional and (2) the data is high-dimensional.

Exercise 2: Bayes Decision Boundaries $(15 + 15$ P$)$

One might speculate that, in some cases, the generated data $p(x|\omega_1)$ and $p(x|\omega_2)$ is of no use to improve the accuracy of a classifier, in which case one should only rely on prior class probabilities $P(\omega_1)$ and $P(\omega_2)$ assumed here to be strictly positive.

For the first part of this exercise, we assume that the data for each class is generated by the univariate Laplacian probability distributions:

$$p(x|\omega_1) = \frac{1}{2\sigma} \exp\Big(-\frac{|x-\mu|}{\sigma}\Big) \quad \text{and} \quad p(x|\omega_2) = \frac{1}{2\sigma} \exp\Big(-\frac{|x+\mu|}{\sigma}\Big).$$

where $\mu, \sigma > 0$.

(a) *Determine* for which values of $P(\omega_1), P(\omega_2), \mu, \sigma$ the optimal decision is to always predict the first class (i.e. under which conditions $P(\text{error}|x) = P(\omega_2|x) \;\; \forall\, x \in \mathbb{R}$).

(b) *Repeat* the exercise for the case where the data for each class is generated by the univariate Gaussian probability distributions:

$$p(x|\omega_1) = \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big) \quad \text{and} \quad p(x|\omega_2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{(x+\mu)^2}{2\sigma^2}\Big).$$

where $\mu, \sigma > 0$.

Exercise 3: Programming (40 P)

Download the programming files on ISIS and follow the instructions.

sheet01-programming

February 15, 2022

1 Programming Sheet 1: Bayes Decision Theory (40 P)

In this exercise sheet, we will apply Bayes decision theory in the context of small two-dimensional problems. For this, we will make use of 3D plotting. We introduce below the basics for constructing these plots in Python/Matplotlib.

1.0.1 The function `numpy.meshgrid`

To plot two-dimensional functions, we first need to discretize the two-dimensional input space. One basic function for this purpose is `numpy.meshgrid`. The following code creates a discrete grid of the rectangular surface $[0, 4] \times [0, 3]$. The function `numpy.meshgrid` takes the discretized intervals as input, and returns two arrays of size corresponding to the discretized surface (i.e. the grid) and containing the X and Y-coordinates respectively.

[ ]:
```
import numpy as np
from matplotlib import pyplot as plt
X, Y = np.meshgrid([0, 1, 2, 3, 4], [0, 1, 2, 3])
print(X)
print(Y)
```

```
[[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]
[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]]
```

Note that we can iterate over the elements of the grid by zipping the two arrays `X` and `Y` containing each coordinate. The function `numpy.flatten` converts the 2D arrays to one-dimensional arrays, that can then be iterated element-wise.

[ ]:
```
print(list(zip(X.flatten(), Y.flatten())))
```

```
[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1),
(0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3)]
```

1.0.2 3D-Plotting

As an example, we would like to plot the L2-norm function $f(x, y) = \sqrt{x^2 + y^2}$ on the subspace $x, y \in [-4, 4]$. First, we create a meshgrid with appropriate size:

```
[ ]: R = np.arange(-4, 4+1e-9, 0.1)
     X, Y = np.meshgrid(R, R)
     print(X.shape,Y.shape)
```

```
(81, 81) (81, 81)
```

Here, we have used a discretization with small increments of 0.1 in order to produce a plot with better resolution. The resulting meshgrid has size (81x81), that is, approximately 6400 points. The function f needs to be evaluated at each of these points. This is achieved by applying element-wise operations on the arrays of the meshgrid. The norm at each point of the grid is therefore computed as:

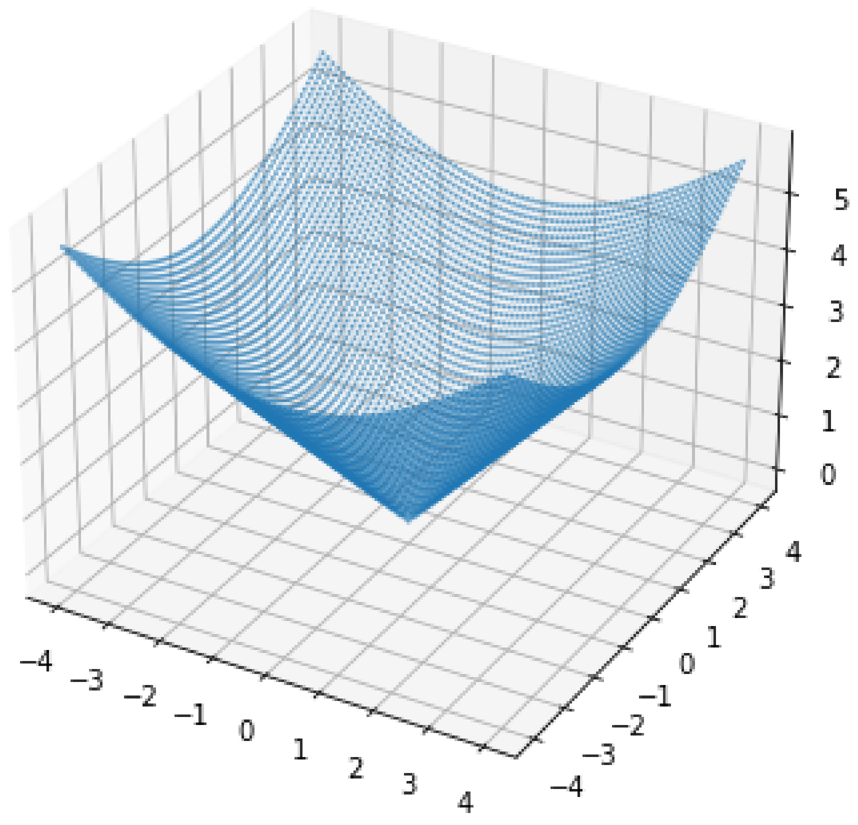
```
[ ]: F = (X**2+ Y**2)**.5
     print(F.shape)
```

```
(81, 81)
```

The resulting function values are of same size as the meshgrid. Taking `X,Y,F` jointly results in a list of approximately 6400 triplets representing the x-, y-, and z-coordinates in the three-dimensional space where the function should be plotted. The 3d-plot can now be constructed easily by means of the function `scatter` of `matplotlib.pyplot`.

```
[ ]: fig = plt.figure(figsize=(10, 6))
     ax = plt.axes(projection='3d')
     ax.scatter(X, Y, F, s=1, alpha=0.5)
```

```
[ ]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x10b66c580>
```



The parameter `s` and `alpha` control the size and the transparency of each data point. Other 3d plotting variants exist (e.g. surface plots), however, the scatter plot is the simplest approach at least conceptually. Having introduced how to easily plot 3D functions in Python, we can now analyze two-dimensional probability distributions with this same tool.

1.1 Exercise 1: Gaussian distributions (5+5+5 P)

Using the technique introduced above, we would like to plot a normal Gaussian probability distribution with mean vector $\mu = (0, 0)$, and covariance matrix $\Sigma = I$ also known as standard normal distribution. We consider the same discretization as above (i.e. a grid from -4 to 4 using step size 0.1). For two-dimensional input spaces, the standard normal distribution is given by:

$$p(x, y) = \frac{1}{2\pi} e^{-0.5(x^2+y^2)}.$$

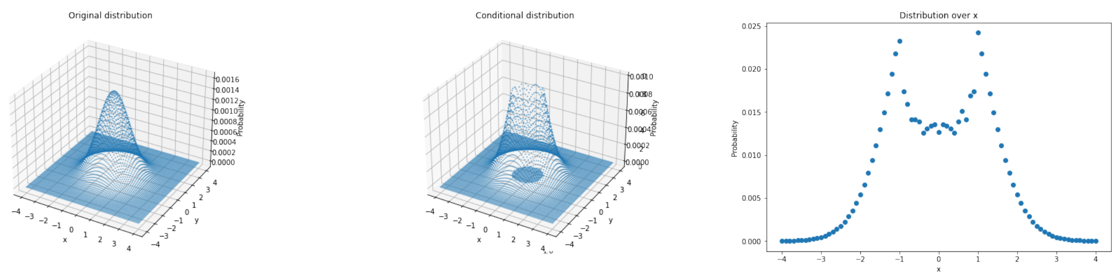
This distribution sums to 1 when integrated over $\mathbb{R}^2$. However, it does not sum to 1 when summing over the discretized space (i.e. the grid). Instead, we can work with a discretized Gaussian-like distribution:

$$P(x, y) = \frac{1}{Z} e^{-0.5(x^2+y^2)} \qquad \text{with} \quad Z = \sum_{x,y} e^{-0.5(x^2+y^2)}$$

where the sum runs over the whole discretized space.

- **Compute the distribution $P(x, y)$, and plot it.**
- **Compute the conditional distribution $Q(x, y) = P((x, y) | \sqrt{x^2 + y^2} \geq 1)$, and plot it.**
- **Marginalize the conditioned distribution $Q(x, y)$ over y, and plot the resulting distribution $Q(x)$.**

[ ]:
```
def P(x, y, mu=0, cov=np.eye(2)):
    exponential = np.exp(-.5 * ((x-mu)**2/cov[0, 0] + (y-mu)**2/cov[1, 1]))
    Z = exponential.sum()
    return 1 / Z * exponential



def Q(x, y, thresh=1):
    mask = (x**2 + y**2) >= thresh
    return P(x, y) * mask



P_discrete = P(X, Y)
Q_discrete = Q(X, Y, thresh=1)
Q_over_x = Q_discrete.sum(0)



fig = plt.figure(figsize=(30, 6))
ax = fig.add_subplot(1, 3, 1, projection='3d')
ax.scatter(X, Y, P_discrete, s=1, alpha=0.5)
ax.set(title="Original distribution", xlabel="x", ylabel="y",␣
 ↪zlabel="Probability")

ax = fig.add_subplot(1, 3, 2, projection='3d')
ax = plt.axes(projection='3d')
ax.scatter(X, Y, Q_discrete, s=1, alpha=0.5)
ax.set(title="Conditional distribution", xlabel="x", ylabel="y",␣
 ↪zlabel="Probability")

ax = fig.add_subplot(1, 3, 3)
ax.scatter(R, Q_over_x)
ax.set(title="Distribution over x", xlabel="x", ylabel="Probability");
```



1.2 Exercise 2: Bayesian Classification (5+5+5 P)

Let the two coordinates x and y be now representated as a two-dimensional vector x. We consider two classes ω_1 and ω_2 with data-generating Gaussian distributions $p(x|\omega_1)$ and $p(x|\omega_2)$ of mean vectors

$$\mu_1 = (-0.5, -0.5) \quad \text{and} \quad \mu_2 = (0.5, 0.5)$$

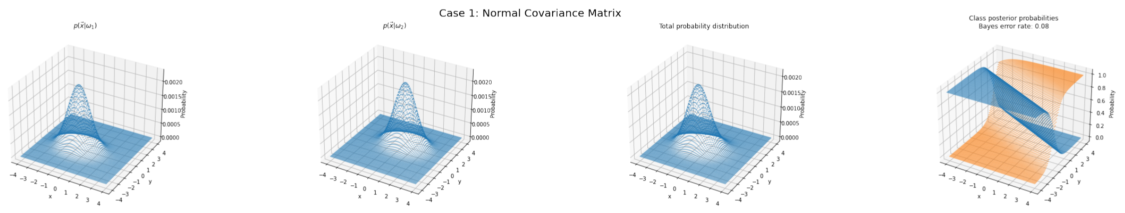
respectively, and same covariance matrix

$$\Sigma = \begin{pmatrix} 1.0 & 0 \\ 0 & 0.5 \end{pmatrix}.$$

Classes occur with probability $P(\omega_1) = 0.9$ and $P(\omega_2) = 0.1$. Analysis tells us that in such scenario, the optimal decision boundary between the two classes should be linear. We would like to verify this computationally by applying Bayes decision theory on grid-like discretized distributions.

- ** Using the same grid as in Exercise 1, discretize the two data-generating distributions $p(x|\omega_1)$ and $p(x|\omega_2)$ (i.e. create discrete distributions $P(x|\omega_1)$ and $P(x|\omega_2)$ on the grid), and plot them with different colors.**
- **From these distributions, compute the total probability distribution $P(x) = \sum_{c\in\{1,2\}} P(x|\omega_c) \cdot P(\omega_c)$, and plot it.**
- **Compute and plot the class posterior probabilities $P(\omega_1|x)$ and $P(\omega_2|x)$, and print the Bayes error rate for the discretized case.**

[ ]:
```
cov = np.array([[1, 0], [0, .5]])

P_disc_1 = P(X, Y, -.5, cov=cov)
P_disc_2 = P(X, Y, .5, cov=cov)

total_prob_dist = P_disc_1 * .9 + P_disc_2 * .1

P_disc_1_normed = P_disc_1 * .9 / total_prob_dist
P_disc_2_normed = P_disc_2 * .1 / total_prob_dist

def compute_bayes_error(p_x, P1, P2):
    bayes_error_x = p_x * np.minimum(P1, P2)
    bayes_error_total = bayes_error_x.sum()
    return bayes_error_total

bayes_error = compute_bayes_error(total_prob_dist, P_disc_1_normed,␣
 ↪P_disc_2_normed)

fig = plt.figure(figsize=(40, 6))
ax = fig.add_subplot(1, 4, 1, projection='3d')
ax.scatter(X, Y, P_disc_1, s=1, alpha=0.5)
ax.set(title=r"$p(\vec{x}|\omega_1$)", xlabel="x", ylabel="y",␣
 ↪zlabel="Probability")

ax = fig.add_subplot(1, 4, 2, projection='3d')
ax.scatter(X, Y, P_disc_2, s=1, alpha=0.5)
ax.set(title=r"$p(\vec{x}|\omega_2$)", xlabel="x", ylabel="y",␣
 ↪zlabel="Probability")

ax = fig.add_subplot(1, 4, 3, projection='3d')
ax.scatter(X, Y, total_prob_dist, s=1, alpha=0.5)
ax.set(title="Total probability distribution", xlabel="x", ylabel="y",␣
 ↪zlabel="Probability")

ax = fig.add_subplot(1, 4, 4, projection='3d')
ax.scatter(X, Y, P_disc_1_normed, s=1, alpha=0.5)
ax.scatter(X, Y, P_disc_2_normed, s=1, alpha=0.5)
ax.set(title=f"Class posterior probabilities\nBayes error rate: {bayes_error:.
 ↪2f}", xlabel="x", ylabel="y", zlabel="Probability")

fig.suptitle("Case 1: Normal Covariance Matrix", fontsize=20);
```

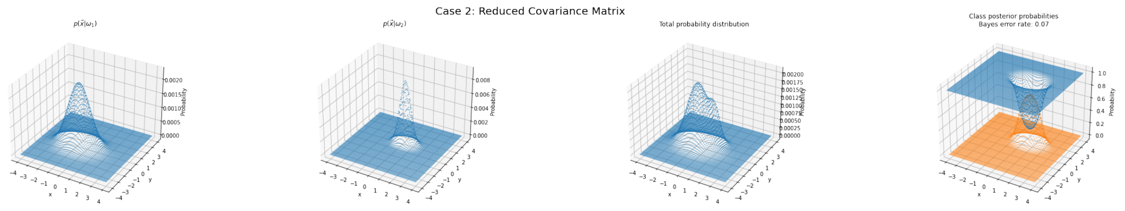


1.3 Exercise 3: Reducing the Variance (5+5 P)

Suppose that the data generating distribution for the second class changes to produce samples much closer to the mean. This variance reduction for the second class is implemented by keeping the first covariance the same (i.e. $\Sigma_1 = \Sigma$) and dividing the second covariance matrix by 4 (i.e. $\Sigma_2 = \Sigma/4$). For this new set of parameters, we can perform the same analysis as in Exercise 2.

- **Plot the new class posterior probabilities $P(\omega_1|x)$ and $P(\omega_2|x)$ associated to the new covariance matrices, and print the new Bayes error rate.**

```
[ ]: P_disc_2 = P(X, Y, .5, cov=cov/4)

     total_prob_dist = P_disc_1 * .9 + P_disc_2 * .1

     P_disc_1_normed = P_disc_1 * .9 / total_prob_dist
     P_disc_2_normed = P_disc_2 * .1 / total_prob_dist

     bayes_error = compute_bayes_error(total_prob_dist, P_disc_1_normed,␣
      ↪P_disc_2_normed)

     fig = plt.figure(figsize=(40, 6))
     ax = fig.add_subplot(1, 4, 1, projection='3d')
     ax.scatter(X, Y, P_disc_1, s=1, alpha=0.5)
     ax.set(title=r"$p(\vec{x}|\omega_1$)", xlabel="x", ylabel="y",␣
      ↪zlabel="Probability")

     ax = fig.add_subplot(1, 4, 2, projection='3d')
     ax.scatter(X, Y, P_disc_2, s=1, alpha=0.5)
     ax.set(title=r"$p(\vec{x}|\omega_2$)", xlabel="x", ylabel="y",␣
      ↪zlabel="Probability")

     ax = fig.add_subplot(1, 4, 3, projection='3d')
     ax.scatter(X, Y, total_prob_dist, s=1, alpha=0.5)
     ax.set(title="Total probability distribution", xlabel="x", ylabel="y",␣
      ↪zlabel="Probability")

     ax = fig.add_subplot(1, 4, 4, projection='3d')
     ax.scatter(X, Y, P_disc_1_normed, s=1, alpha=0.5)
     ax.scatter(X, Y, P_disc_2_normed, s=1, alpha=0.5)
     ax.set(title=f"Class posterior probabilities\nBayes error rate: {bayes_error:.
      ↪2f}", xlabel="x", ylabel="y", zlabel="Probability")

     fig.suptitle("Case 2: Reduced Covariance Matrix", fontsize=20);
```



Intuition tells us that by variance reduction and resulting concentration of generated data for class 2 in a smaller region of the input space, it should be easier to predict class 2 with certainty at this location. Paradoxally, in this new "dense" setting, we observe that class 2 does not reach full certainty anywhere in the input space, whereas it did in the previous exercise.

- **Explain this paradox.**

1.3.1 Answer:

First case: Class 2 reached full certainty only where class 1 was impossible. In general, very few observations will occur in that space.

Second case: Class 2 does no longer reach full certainty because due to its reduced variance it covers less space in locations with small probability. Now it is much more likely to be observed at its maximum. However, since its prior probability of 10% is very low, class 1 is still more likely at the maximum of class 2.

Machine Learning 1

Week 1

$\boxed{1}$ a) $P(\text{error}) = \int P(\text{error} \mid \vec{x}) \cdot p(\vec{x}) \cdot d\vec{x}$

$= \int \min[P(\omega_1 \mid \vec{x}), P(\omega_2 \mid \vec{x})] \cdot p(\vec{x}) \cdot d\vec{x}$

1) $= \int M_{-\infty}(P(\omega_1 \mid \vec{x}), P(\omega_2 \mid \vec{x}))\, p(\vec{x})\, d\vec{x}$

2) $\leq \int M_{-1}(P(\omega_1 \mid \vec{x}), P(\omega_2 \mid \vec{x}))\, p(\vec{x})\, d\vec{x}$

3) $= \int \frac{2}{\frac{1}{P(\omega_1 \mid \vec{x})} + \frac{1}{P(\omega_2 \mid \vec{x})}}\, p(\vec{x})\, d\vec{x}$

with 1) $M_p(x_1, \dots, x_n) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$

2) $M_p \leq M_q$ if $p < q$

3) $M_{-1}(P(\omega_1 \mid \vec{x}), P(\omega_2 \mid \vec{x})) = \left(\frac{1}{2} P(\omega_1 \mid \vec{x})^{-1} + \frac{1}{2} P(\omega_2 \mid x)^{-1}\right)^{-1}$

$= \frac{1}{\frac{1}{2}\frac{1}{P(\omega_1 \mid \vec{x})} + \frac{1}{2}\frac{1}{P(\omega_2 \mid \vec{x})}} = \frac{2}{\frac{1}{P(\omega_1 \mid \vec{x})} + \frac{1}{P(\omega_2 \mid \vec{x})}}$

b)

[illegible]

$$P(\text{error}) \le \int \frac{2}{\frac{1}{P(\omega_1|\vec{x})} + \frac{1}{P(\omega_2|\vec{x})}} \, p(\vec{x})\, d\vec{x}$$

Bayes rule:

$P(\omega_j|\vec{x}) = \frac{p(\vec{x}|\omega_j)P(\omega_j)}{p(\vec{x})}$

$$= \int \frac{2}{\frac{1}{p(\vec{x}|\omega_1)P_1} + \frac{1}{p(\vec{x}|\omega_2)P_2}} \, d\vec{x}$$

$P(\omega_j) = P_j$

$p(\vec{x}|\omega_1) = \frac{\pi^{-1}}{1+(x-\mu)^2}$

$p(\vec{x}|\omega_2) = \frac{\pi^{-1}}{1+(x+\mu)^2}$

$$= \int \frac{2}{\frac{1}{\frac{\pi^{-1}P_1}{1+(x-\mu)^2}} + \frac{1}{\frac{\pi^{-1}P_2}{1+(x+\mu)^2}}} \, d\vec{x}$$

$$= \int \frac{2P_1P_2\pi^{-1}}{P_2(1+(x-\mu)^2) + P_1(1+(x+\mu)^2)}$$

$$= 2P_1P_2\pi^{-1} \int \frac{1}{\underbrace{1\vec{x}}_{a} + \underbrace{2\mu(P_1-P_2)\vec{x}}_{b} + \underbrace{1+\mu^2}_{c}} \, d\vec{x}$$

$\int \frac{1}{ax^2+bx+c}\, dx = \frac{2\pi}{\sqrt{4ac-b^2}}$

In our case

$(2\mu(P_1-P_2))^2 < 4(1+\mu^2)$

because $(P_1-P_2)^2 < 1$

$$= 2P_1P_2 \frac{2\pi}{\sqrt{4+4\mu^2-4\mu^2(P_1-P_2)^2}}$$

$$= 2P_1P_2 \frac{1}{\sqrt{4+16\mu^2P_1P_2}}$$

$$= \frac{2P_1P_2}{\sqrt{1+4\mu^2P_1P_2}}$$

c) 1) integrate the original error function $P(\text{error}|\vec{x})$ numerically for low dimensional data

2) Create discriminants $g_1(x), g_2(x)$ supporting an optimal classifier and sample hierarchically $\omega_j \sim [P(\omega_1), P(\omega_2)]$ and $P(\vec{x}|\omega_j)$ and compute the average.

$\boxed{2}$ a) For Bayes optimal decision boundary we need two discriminants

$$g_1(x) = -\frac{|x-\mu|}{\sigma} + \log P(\omega_1)$$

$$g_2(x) = -\frac{|x+\mu|}{\sigma} + \log P(\omega_2)$$

Next, we need to check for which parameter values and all x $\forall_x : g_1(x) \geq g_2(x)$ holds. We have

$$x \leq -\mu \quad \Rightarrow \quad 2\mu \leq \sigma \cdot [\log P(\omega_1) - \log P(\omega_2)]$$
$$-\mu < x < \mu \quad \Rightarrow \quad 2x \leq \sigma [\log P(\omega_1) - \log P(\omega_2)]$$
$$x \geq \mu \quad \Rightarrow \quad -2\mu \leq \sigma [\log P(\omega_1) - \log P(\omega_2)]$$

From this set of equations we can infer the parameter for ω_1 prediction

$$\left\{ (P(\omega_1), P(\omega_2), \mu, \sigma) \,\middle|\, \log \frac{P(\omega_1)}{P(\omega_2)} \geq \frac{2\mu}{\sigma} \right\}$$

b) The new pair of discriminants is

$$g_1(x) = -\frac{(x-\mu)^2}{2\sigma^2} + \log P(\omega_1)$$

$$g_2(x) = -\frac{(x+\mu)^2}{2\sigma^2} + \log P(\omega_2)$$

The inequality $g_1(x) \geq g_2(x)$ becomes

$$-2x\mu \leq \sigma [\log P(\omega_1) - \log P(\omega_2)]$$

There is no set of parameters that always predicts class ω_1.