Machine Learning 1 WS18/19

Submission for Exercise Sheet 1

Hendrik Makait 384968
Michael Hoppe 362514
Wai Tang Victor Chan 406094
Rudi Poepsel Lemaitre 373017
Jonas Piotrowski 399334
Aki Saksala 399293

Exercise Sheet 1 **Machine Learning 1**

Exercise 1:

a)

$$P(error) = \int P(error|x) * p(x)\, dx$$

$$P(error) = \int \min[P(\omega_1|x), P(\omega_2|x)] * p(x)\, dx$$

With lose of generosity, we assume that $P(\omega_1|x) \leq P(\omega_2|x)$ in other words

$P(\omega_1|x) = P(error|x)$

Hence:

$$P(\omega_1|x) \geq P(\omega_2|x)$$

$$\frac{1}{P(\omega_1|x)} \geq \frac{1}{P(\omega_2|x)}$$

$$\frac{2}{P(\omega_1|x)} \geq \frac{1}{P(\omega_2|x)} + \frac{1}{P(\omega_1|x)}$$

$$P(error|x) = P(\omega_1|x) \leq \frac{2}{\frac{1}{P(\omega_2|x)} + \frac{1}{P(\omega_1|x)}}$$

$$P(error) \leq \int \frac{2}{\frac{1}{P(\omega_2|x)} + \frac{1}{P(\omega_1|x)}} * p(x)\, dx$$

b)

Consider that $P(\omega_i|x) = \frac{P(x|\omega_i)*P(\omega_i)}{P(x)}$ for $i = 1,2$

So

$$P(error) = \int \frac{2}{\frac{P(x)}{P(x|\omega_1) * P(\omega_1)} + \frac{P(x)}{P(x|\omega_2) * P(\omega_2)}} * p(x)\, dx$$

$$= \int \frac{2}{\frac{\pi(1+(x-\mu)^2)}{P(\omega_1)} + \frac{\pi(1+(x+\mu)^2)}{P(\omega_2)}} dx$$

$$= 2 * P(\omega_1) * P(\omega_2) * \frac{1}{\pi} \int \frac{1}{x^2 + 2x\mu\big(P(\omega_1) - P(\omega_2)\big) + (\mu^2 + 1)} dx$$

First, we need to check in order to be able to use the property to solve the integral

$$\Big(2\mu\big(P(\omega_1) - P(\omega_2)\big)\Big)^2 - 4 * (\mu^2 + 1) < 0$$

$$4(\mu^2(P(\omega_1)^2 - 2 * P(\omega_1) * P(\omega_2) + P(\omega_2)^2 - 1) - 1) < 0$$

$$4\left(\mu^2\left(\big(P(\omega_1) + P(\omega_2)\big)^2 - 4 * P(\omega_1) * P(\omega_2) - 1\right) - 1\right) < 0$$

$$4\big(\mu^2\big(-4 * P(\omega_1) * P(\omega_2)\big) - 1\big) < 0$$

$$-4(4 * \mu^2 * P(\omega_1) * P(\omega_2) + 1) < 0$$

Now using the identity

$$P(error) = 2 * P(\omega_1) * P(\omega_2) * \frac{1}{\pi} * \frac{2 * \pi}{2\sqrt{4 * \mu^2 * P(\omega_1) * P(\omega_2) + 1}}$$

$$P(error) = \frac{2 * P(\omega_1) * P(\omega_2)}{\sqrt{4 * \mu^2 * P(\omega_1) * P(\omega_2) + 1}}$$

c)

If no analytical integration can help to get an upper bound, we will just resort to numerical integration and use it to directly compute the error.

For lower dimension case, classical method such as the trapezoidal rules will suffice, especially since the min functions is piecewise continuous. And for low dimension case, the number of computation required will not be so large that it cannot be handled.

For higher dimension case, we can use a recently developed method called the Randomized lattice rules, which can converge reasonably fast even with high dimension and so solve the problem of the dimensionality curse.

Ex. 2: Given: $p(x|\omega_1) = \frac{1}{2\sigma} \exp\left(-\frac{|x-\mu|}{\sigma}\right)$,

$$p(x|\omega_2) = \frac{1}{2\sigma} \exp\left(-\frac{|x+\mu|}{\sigma}\right) \quad \text{with} \quad \mu, \sigma > 0$$

a)

The optimal decision boundary is given by

$P(\omega_1|x) = P(\omega_2|x)$. Solve for x:

$$P(\omega_1|x) = P(\omega_2|x)$$

Bayes $\Leftrightarrow$ $\frac{p(x|\omega_1)P(\omega_1)}{p(x)} = \frac{p(x|\omega_2)P(\omega_2)}{p(x)}$

$\cdot p(x)$ $\Leftrightarrow$

$$\Leftrightarrow p(x|\omega_1)P(\omega_1) = p(x|\omega_2)P(\omega_2)$$

def. $\Leftrightarrow \frac{1}{2\sigma} \exp\left(-\frac{|x-\mu|}{\sigma}\right) P(\omega_1) = \frac{1}{2\sigma} \exp\left(-\frac{|x+\mu|}{\sigma}\right) P(\omega_2)$

$\cdot 2\sigma$ $\Leftrightarrow \exp\left(-\frac{|x-\mu|}{\sigma}\right) P(\omega_1) = \exp\left(-\frac{|x+\mu|}{\sigma}\right) P(\omega_2)$

ln $\Leftrightarrow -\frac{|x-\mu|}{\sigma} + \ln P(\omega_1) = -\frac{|x+\mu|}{\sigma} + \ln P(\omega_2)$

$$\Leftrightarrow \ln P(\omega_1) - \ln P(\omega_2) = \frac{|x-\mu| - |x+\mu|}{\sigma}$$

$$\Leftrightarrow \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = \frac{|x-\mu| - |x+\mu|}{\sigma}$$

$\cdot \sigma$ $\Leftrightarrow \sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = |x-\mu| - |x+\mu|$

Case 1: $x-\mu \geq 0 \wedge x+\mu \geq 0 \Leftrightarrow x \geq \mu \wedge x \geq -\mu \overset{\mu>0}{\Leftrightarrow} x \geq \mu$

$$\sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = x-\mu-(x+\mu) = -2\mu$$

Case 2: $x-\mu<0 \wedge x+\mu \geq 0 \Leftrightarrow x<\mu \wedge x \geq -\mu \overset{\mu>0}{\Leftrightarrow} -\mu \leq x < \mu$

$$\sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = -(x-\mu)-(x+\mu) = -2x$$

$$\overset{(\cdot -\frac{1}{2})}{\Leftrightarrow} -\frac{\sigma}{2} \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = x$$

Case 3: $x-\mu<0 \wedge x+\mu<0 \Leftrightarrow x<\mu \wedge x<-\mu \overset{\mu>0}{\Leftrightarrow} x<-\mu$

$$\sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = -(x-\mu)-(-1)(x+\mu) = -(x-\mu)+(x+\mu) = 2\mu$$

b) $P(\text{error}|x) = \min\left(P(\omega_1|x), P(\omega_2|x)\right)$

$\Rightarrow$ For $P(\text{error}|x) = P(\omega_2|x)$ to hold $\forall x \in \mathbb{R}$, it must also hold that $P(\omega_1|x) > P(\omega_2|x)\ \forall x \in \mathbb{R}$.

$$P(\omega_1|x) > P(\omega_2|x)$$

$$\Leftrightarrow \quad \frac{p(x|\omega_1)P(\omega_1)}{p(x)} > \frac{p(x|\omega_2)P(\omega_2)}{p(x)}$$

def. $p(x) > 0$

$$\Leftrightarrow \quad \frac{1}{2\sigma}\exp\left(-\frac{|x-\mu|}{\sigma}\right)P(\omega_1) > \frac{1}{2\sigma}\exp\left(-\frac{|x+\mu|}{\sigma}\right)P(\omega_2)$$

$2\sigma > 0$, ln monot. $\uparrow$

$$\Leftrightarrow \quad -\frac{|x-\mu|}{\sigma} + \ln P(\omega_1) > -\frac{|x+\mu|}{\sigma} + \ln P(\omega_2)$$

$$\Leftrightarrow \quad \ln P(\omega_1) - \ln P(\omega_2) > \frac{|x-\mu| - |x+\mu|}{\sigma}$$

$\sigma > 0$

$$\Leftrightarrow \quad \sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > \underbrace{|x-\mu| - |x+\mu|}_{\text{3 cases:}}$$

Case 1: $x-\mu \geq 0 \wedge x+\mu \geq 0 \Leftrightarrow x \geq \mu$

$$x-\mu-(x+\mu) = -2\mu$$

Case 2: $x-\mu < 0 \wedge x+\mu \geq 0 \Leftrightarrow -\mu \leq x < \mu$

cond. $\mu > 0$

$$-(x-\mu)-(x+\mu) = -2x \leq -2(-\mu) = 2\mu$$

Case 3: $x-\mu < 0 \wedge x+\mu < 0 \Leftrightarrow x < -\mu$

$$-(x-\mu)-(-1)(x+\mu) = 2\mu$$

From case 1, case 2, case 3 and $\mu > 0$ follows: $|x-\mu| - |x+\mu| \leq 2\mu$ $\forall x \in \mathbb{R}$.

Thus $P(\text{error}|x) = P(\omega_2|x) \quad \forall x \in \mathbb{R}$

for all $\sigma, \mu, P(\omega_1), P(\omega_2)$ s.t.

$$\sigma \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > 2\mu \quad \square$$

c) Given: $p(x|\omega_1) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$,

$p(x|\omega_2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x+\mu)^2}{2\sigma^2}\right)$, $\sigma > 0$

The optimal decision boundary is given by

$P(\omega_1|x) = P(\omega_2|x)$. Solve for x:

$P(\omega_1|x) = P(\omega_2|x)$

Bayes $\Leftrightarrow$ $\frac{p(x|\omega_1)P(\omega_1)}{p(x)} = \frac{p(x|\omega_2)P(\omega_2)}{p(x)}$

$\cdot p(x)$ $\Leftrightarrow$ $p(x|\omega_1)P(\omega_1) = p(x|\omega_2)P(\omega_2)$

def. $\Leftrightarrow$ $\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) P(\omega_1) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) P(\omega_2)$

$\cdot \sigma\sqrt{2\pi}$ $\Leftrightarrow$ $\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) P(\omega_1) = \exp\left(-\frac{(x+\mu)^2}{2\sigma^2}\right) P(\omega_2)$

ln $\Leftrightarrow$ $-\frac{(x-\mu)^2}{2\sigma^2} + \ln P(\omega_1) = -\frac{(x+\mu)^2}{2\sigma^2} + \ln P(\omega_2)$

$\Leftrightarrow$ $\ln P(\omega_1) - \ln P(\omega_2) = \frac{(x-\mu)^2 - (x+\mu)^2}{2\sigma^2}$

$\cdot 2\sigma^2$ $\Leftrightarrow$ $2\sigma^2 \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = x^2 - 2x\mu + \mu^2 - x^2 - 2x\mu - \mu^2$

$\Leftrightarrow$ $2\sigma^2 \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = -4x\mu$

$\cdot\left(-\frac{1}{4\mu}\right)$ $\Leftrightarrow$ $-\frac{\sigma^2}{2\mu} \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) = x$

$$P(\text{error}|x) = \min\left(P(\omega_1|x), P(\omega_2|x)\right)$$

$$\Rightarrow P(\text{error}|x) = P(\omega_2|x) \text{ if } P(\omega_1|x) > P(\omega_2|x):$$

It must hold $\forall x \in \mathbb{R}$:

$$P(\omega_1|x) > P(\omega_2|x)$$

$$\Leftrightarrow \frac{p(x|\omega_1)P(\omega_1)}{p(x)} > \frac{p(x|\omega_2)P(\omega_2)}{p(x)}$$

$$\Leftrightarrow \frac{1}{2\sigma}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) > \frac{1}{2\sigma}\exp\left(-\frac{(x+\mu)^2}{2\sigma^2}\right) P(\omega_2)$$

$P(\omega_1)$

$$\overset{\ln}{\Leftrightarrow} -\frac{(x-\mu)^2}{2\sigma^2} + \ln P(\omega_1) > -\frac{(x+\mu)^2}{2\sigma^2} + \ln P(\omega_2)$$

$$\overset{\sigma>0}{\Leftrightarrow} 2\sigma^2 \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > (x-\mu)^2 - (x+\mu)^2$$

$$\Leftrightarrow 2\sigma^2 \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > -4x\mu$$

$$\Leftrightarrow \frac{\sigma^2}{2} \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > -x\mu$$

For this to hold $\forall x \in \mathbb{R}$, μ must be 0 as $x \in \mathbb{R}$.

Then ~~$P(\text{error}|x)$~~ it must hold that $\frac{\sigma^2}{2} \ln\left(\frac{P(\omega_1)}{P(\omega_2)}\right) > 0$

$$\Leftrightarrow \sigma^2 \ln P(\omega_1) - \sigma^2 \ln P(\omega_2) > 0$$

$$\Leftrightarrow \sigma^2 \ln P(\omega_1) > \sigma^2 \ln P(\omega_2)$$

$$\Leftrightarrow \ln P(\omega_1) > \ln P(\omega_2)$$

$\Leftrightarrow \quad P(\omega_1) > P(\omega_2) \quad \Rightarrow \sigma > 0$, arbitrary

$\Rightarrow P(\text{error} \mid x) = P(\omega_2 \mid x) \quad \forall x \in \mathbb{R}$

for all $\mu, \sigma, P(\omega_1), P(\omega_2)$, s.t. $\mu = 0$, $\sigma > 0$ arbitrary and $P(\omega_1) > P(\omega_2)$ ■

Programming Tasks

October 22, 2018

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.

```
In [53]: import numpy
         X,Y = numpy.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.

```
In [54]: 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
```

1.0.2 3D-Plotting

To enable 3D-plotting, we first need to load some modules in addition to `matplotlib`:

```
In [55]: import matplotlib
         %matplotlib inline
         from matplotlib import pyplot as plt
         plt.rcParams["figure.figsize"] = 12, 9
         from mpl_toolkits.mplot3d import Axes3D
         import math
         from scipy import stats
```

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:

```
In [56]: R = numpy.arange(-4,4+1e-9,0.1)
         X,Y = numpy.meshgrid(R,R)
         print(X.shape,Y.shape)
```

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

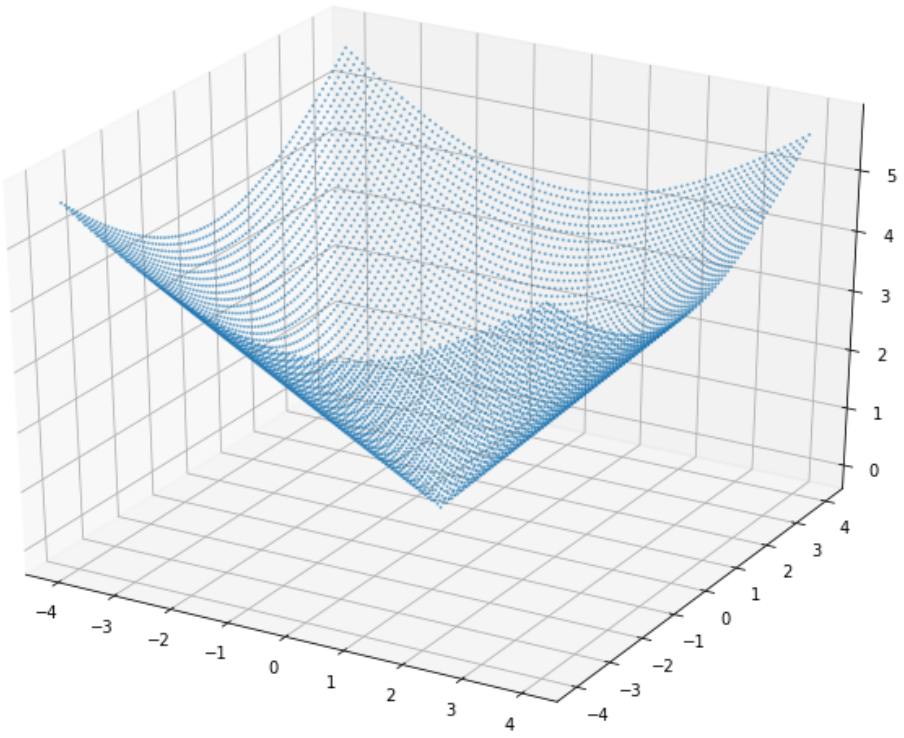
Here, we have used a discretization with small increments of 0.1 in order to produce a plot with better resolution. The resulting meshgrid has size (81x81), that is, approximately 6400 points. The function f needs to be evaluated at each of these points. This is achieved by applying element-wise operations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

```
In [57]: 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`.

```
In [58]: fig = plt.figure()
         ax = plt.axes(projection='3d')
         ax.scatter(X,Y,F,s=1,alpha=0.5)
         plt.show()
```



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:

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

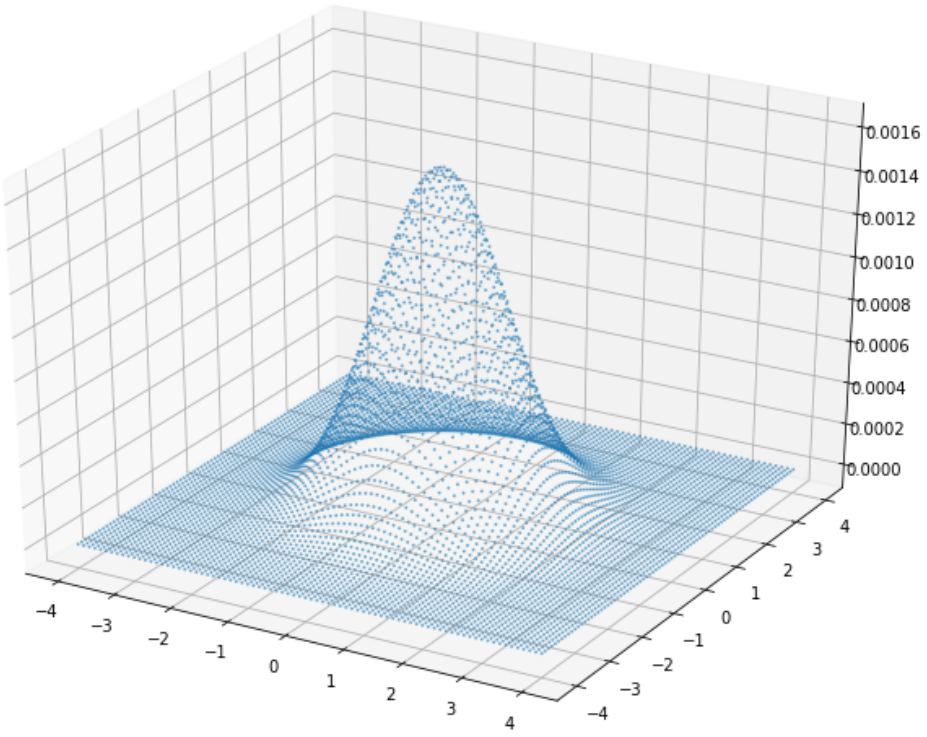
where the sum runs over the whole discretized space.

- **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 $Q(x)$.**

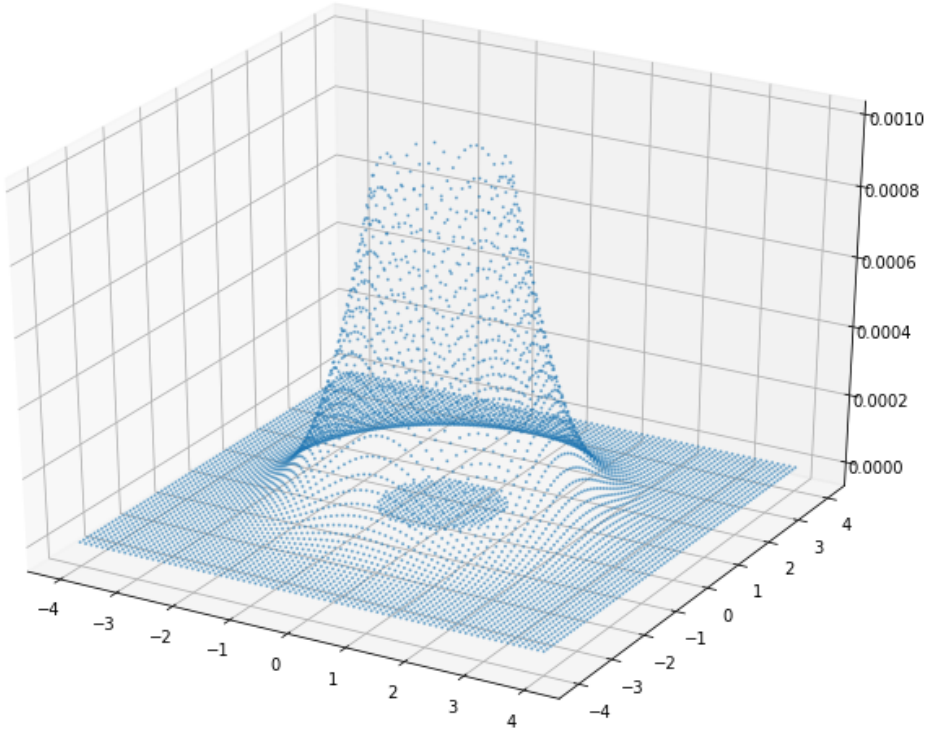
In [59]:
```
### REPLACE BY YOUR CODE
Z = 0.0
points = list(zip(X.flatten(), Y.flatten()))
for p in points:
    Z += math.exp(-0.5 * (p[0]**2 + p[1]**2))

P = numpy.zeros(X.shape)
for r in range(P.shape[0]):
    for c in range(P.shape[1]):
        P[r][c] = (1 / Z) * math.exp(-0.5 * (X[r][c]**2 + Y[r][c]**2))

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P,s=1,alpha=0.5)
###
```

Out[59]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1250d7ad0>



In [60]:
```
### REPLACE BY YOUR CODE
Q = numpy.zeros(X.shape)
for r in range(Q.shape[0]):
    for c in range(Q.shape[1]):
        if math.sqrt(X[r][c]**2 + Y[r][c]**2) >= 1:
            Q[r][c] = (1 / Z) * math.exp(-0.5 * (X[r][c]**2 + Y[r][c]**2))

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X,Y,Q,s=1,alpha=0.5)
###
```

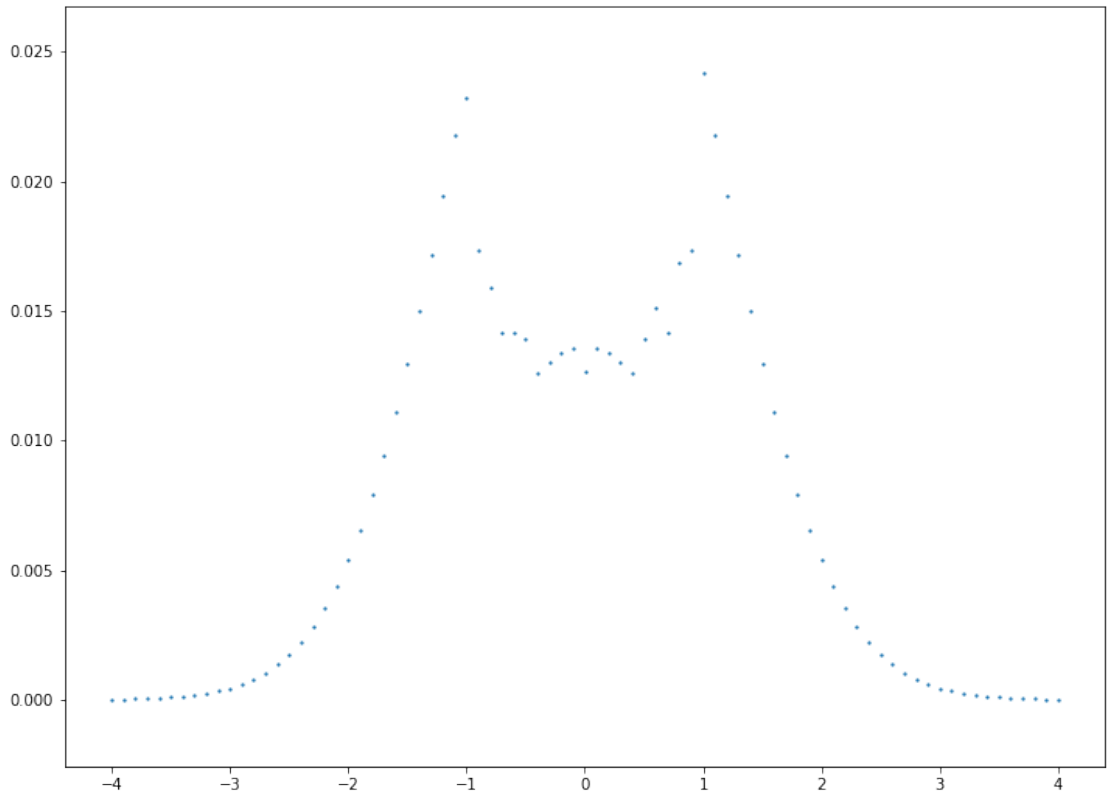
Out[60]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x12452a150>



In [61]:
```
### REPLACE BY YOUR CODE
Q_marginalized = numpy.zeros(X.shape[0])
result = 0
for x in range(X.shape[0]):
    for y in range(X.shape[1]):
        result += Q[x][y]
    Q_marginalized[x] = result
```

```
            result = 0

        fig = plt.figure()
        ax = plt.axes()
        ax.scatter(X[0],Q_marginalized,s=1.5,alpha=1)
        ###
```

```
Out[61]: <matplotlib.collections.PathCollection at 0x127ea09d0>
```



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 $\boldsymbol{x}$. We consider two classes ω_1 and ω_2 with data-generating Gaussian distributions $p(\boldsymbol{x}|\omega_1)$ and $p(\boldsymbol{x}|\omega_2)$ of mean vectors

$$\boldsymbol{\mu}_1 = (-0.5, -0.5) \quad \text{and} \quad \boldsymbol{\mu}_2 = (0.5, 0.5)$$

respectively, and same covariance matrix

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

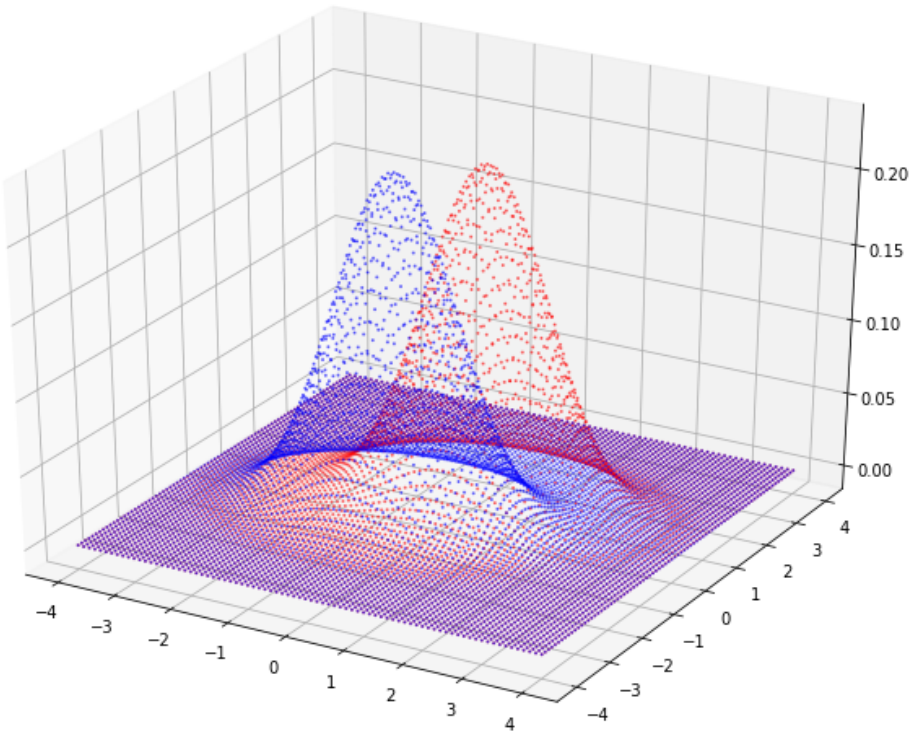
Classes occur with probability $P(\omega_1) = 0.9$ and $P(\omega_2) = 0.1$. Analysis tells us that in such scenario, the optimal decision boundary between the two classes should be linear. We would like to

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

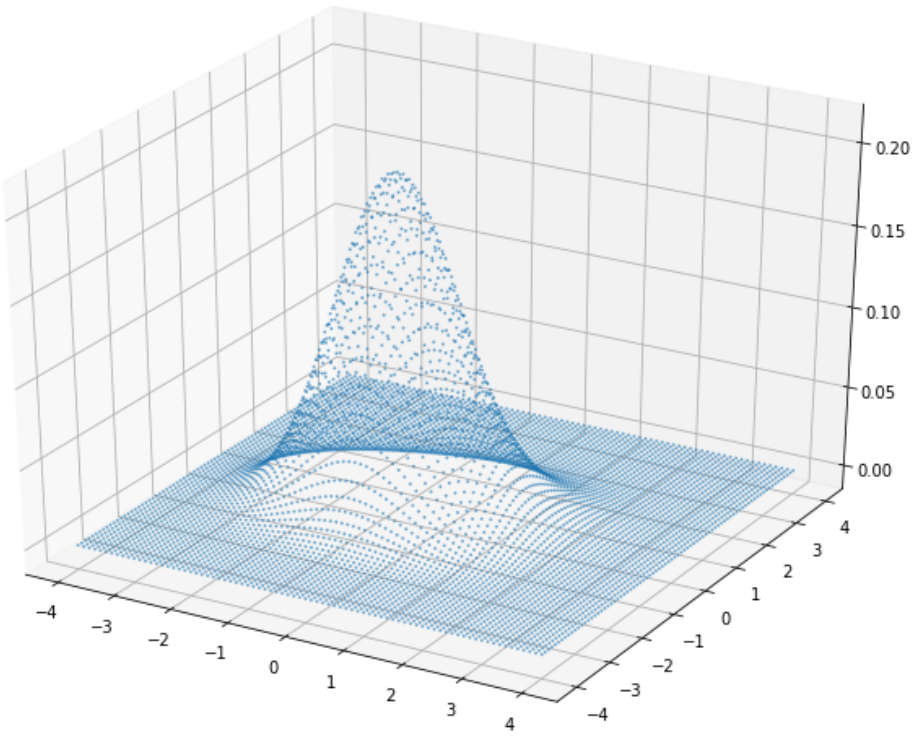
In [62]:
```
### REPLACE BY YOUR CODE
m1 = numpy.array([-0.5, -0.5])
m2 = numpy.array([0.5, 0.5])
covariance = numpy.array([[1.0, 0.0], [0.0, 0.5]])

P1 = stats.multivariate_normal.pdf(points, m1, covariance)
P1 = numpy.reshape(P1, X.shape)

P2 = stats.multivariate_normal.pdf(points, m2, covariance)
P2 = numpy.reshape(P2, X.shape)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P1,s=1,alpha=0.5,color='b')
ax.scatter(X,Y,P2,s=1,alpha=0.5,color='r')
###
```

Out[62]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x128005910>
```



In [63]:
```
### REPLACE BY YOUR CODE
P1_total = P1 * 0.9
P2_total = P2 * 0.1
P_total = P1_total + P2_total

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P_total,s=1,alpha=0.5)
###
```

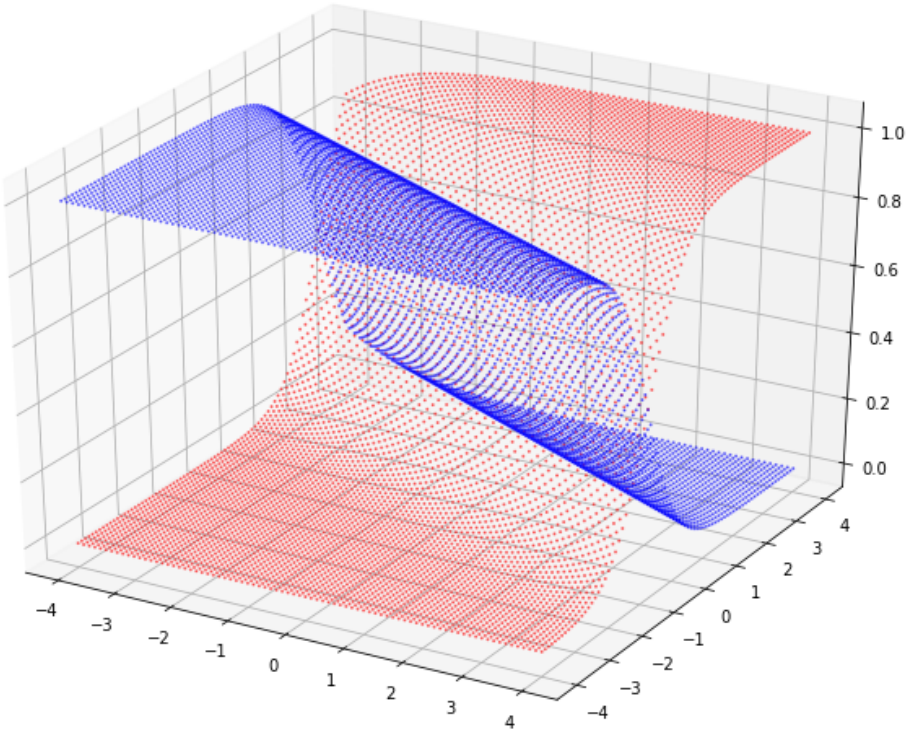
Out[63]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x128005450>
```




```
In [64]: ### REPLACE BY YOUR CODE
         P1_post = P1_total / P_total
         P2_post = P2_total / P_total

         fig = plt.figure()
         ax = plt.axes(projection='3d')
         ax.scatter(X,Y,P1_post,s=1,alpha=0.5,color='b')
         ax.scatter(X,Y,P2_post,s=1,alpha=0.5,color='r')

         bayes_array = list()

         bayes_error_rate = 0
         for x in range(X.shape[0]):
             for y in range(X.shape[1]):
                 if P1_post[x][y] > P2_post[x][y]:
                     single_error_rate = P2_post[x][y]
                 else:
                     single_error_rate = P1_post[x][y]
                 bayes_error_rate += single_error_rate * P_total[x][y] * 0.1 * 0.1

         print(bayes_error_rate)
```

```
        ###
```

```
0.08040553760622328
```



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

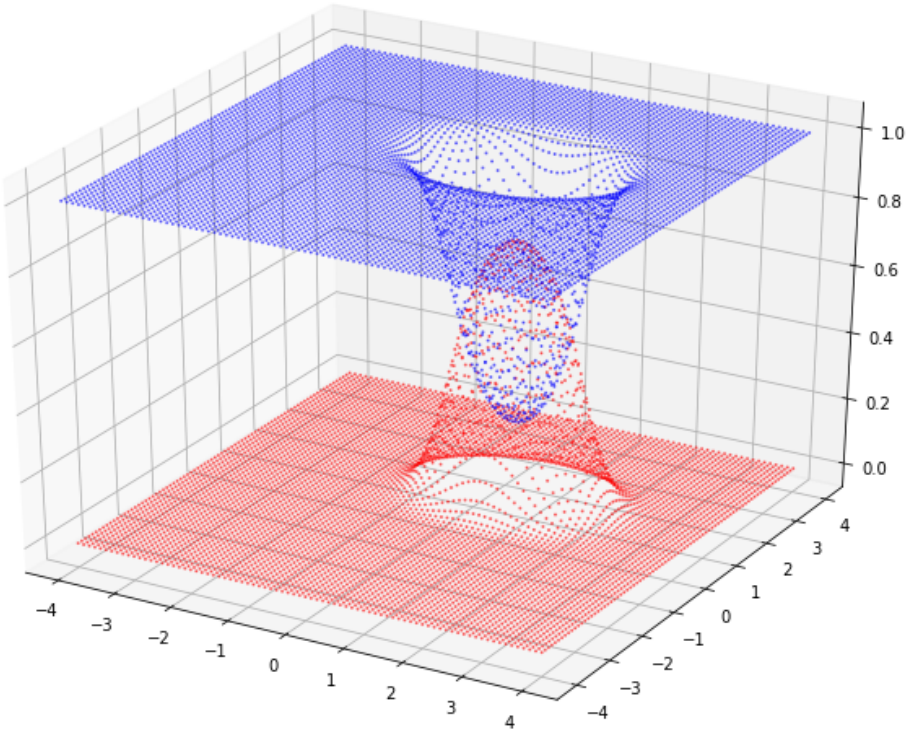
Suppose that the data generating distribution for the second class changes to produce samples much closer to the mean. This variance reduction for the second class is implemented by keeping the first covariance the same (i.e. $\Sigma_1 = \Sigma$) and dividing the second covariance matrix by 4 (i.e. $\Sigma_2 = \Sigma/4$). 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.**

```
In [65]: ### REPLACE BY YOUR CODE
         covariance_new = covariance / 4

         P2_new = stats.multivariate_normal.pdf(points, m2, covariance_new)
         P2_new = numpy.reshape(P2_new, X.shape)

         P2_new_total = P2_new * 0.1
```

```
P_new_total = P1_total + P2_new_total

P1_new_post = P1_total / P_new_total
P2_new_post = P2_new_total / P_new_total

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P1_new_post,s=1,alpha=0.5,color='b')
ax.scatter(X,Y,P2_new_post,s=1,alpha=0.5,color='r')

bayes_array = list()

bayes_error_rate = 0
for x in range(X.shape[0]):
    for y in range(X.shape[1]):
        if P1_new_post[x][y] > P2_new_post[x][y]:
            single_error_rate = P2_new_post[x][y]
        else:
            single_error_rate = P1_new_post[x][y]
        bayes_error_rate += single_error_rate * P_new_total[x][y] * 0.1 * 0.1

print(bayes_error_rate)
###
```

```
0.07290160794820284
```



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.**

Answer:
The variance being smaller in the "dense" setting does not mean that $P(\omega_2|x)$ in this region is higher. The main reason for class 2 not reaching full certainty is that the means of the classes have a relatively short distance to each other, meaning that the distribution of $P(x|\omega_2)$ in the "dense" setting does not reach the areas where $P(x|\omega_1)$ has a low value, but only grows in areas where $P(x|\omega_1)$ is also relatively high. In addition to this, the prior probability $P(\omega_2)$ is much lower than $P(\omega_1)$, which leads to $P(\omega_2|x)$ not reaching full certainty, even at the mean, where $P(x|\omega_2)$ is highest.

In [ ]: