Machine Learning Assignment 1

October 19, 2018

Group Name : XTBQJ

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.

1.1 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 [190]: 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 [191]: 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)]
```

1.2 3D-Plotting

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

```
In [192]: import matplotlib
          %matplotlib inline
          from matplotlib import pyplot as plt
          from mpl_toolkits.mplot3d import Axes3D
```

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:

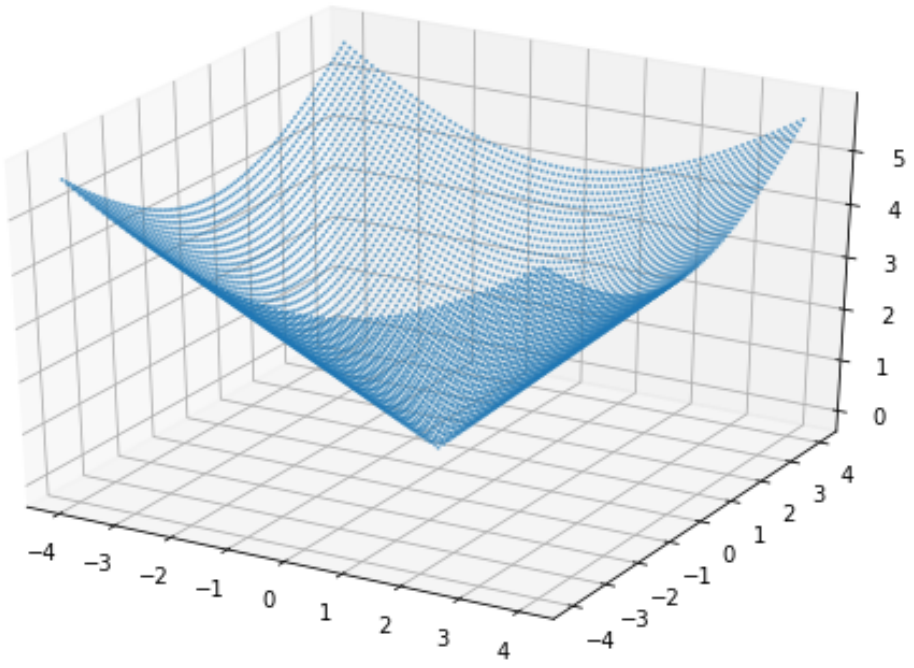
```
In [193]: R = numpy.arange(-4,4+1e-9,0.1)
          X,Y = numpy.meshgrid(R,R)
          print(X.shape,Y.shape)

          (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 [194]: 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 [195]: fig = plt.figure(figsize=(9,6))
          ax = plt.axes(projection='3d')
          ax.scatter(X,Y,F,s=1,alpha=0.5)

Out[195]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x24b089632b0>
```



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.

1.3 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:

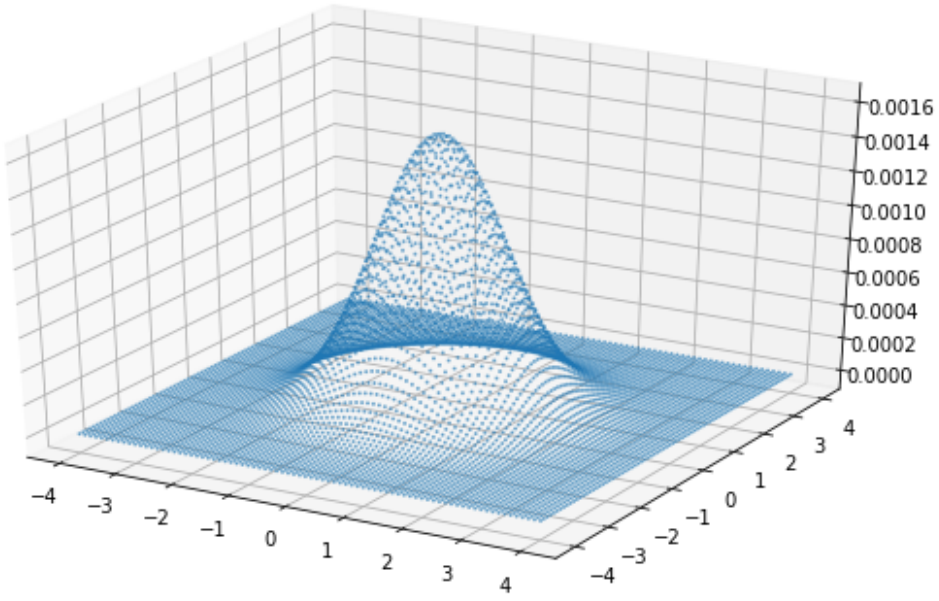
$$P(x, y) = \frac{1}{Z} e^{-0.5(x^2+y^2)} \qquad \text{with} \quad Z = \sum_{x,y} e^{-0.5(x^2+y^2)}$$

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 $Q(x)$.**

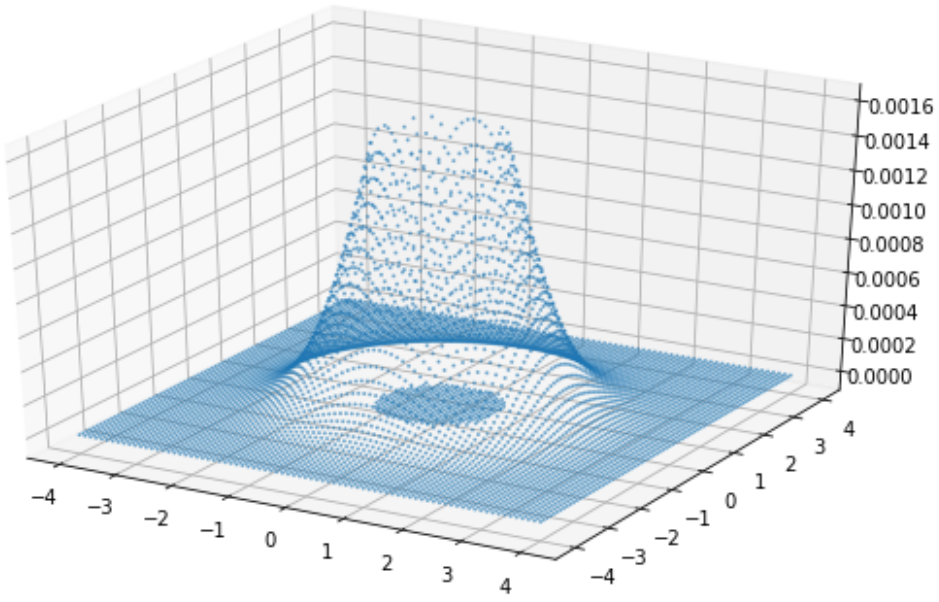
In [196]:
```
### REPLACE BY YOUR CODE
f = numpy.exp(-0.5*(X**2+Y**2))
Z = numpy.sum(f)
P = f/Z
fig = plt.figure(figsize=(10,6))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P,s=1,alpha=0.5)
###
```

Out[196]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x24b08b5f780>

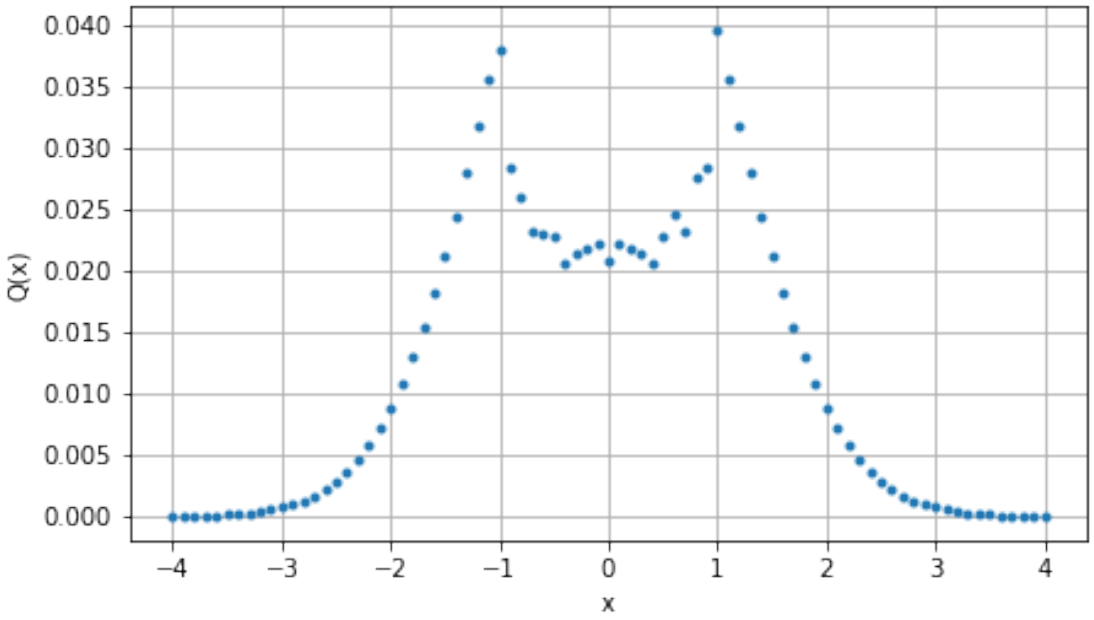


In [197]:
```
### REPLACE BY YOUR CODE
Q = numpy.zeros(P.shape)
index = numpy.where((F>=1))
Q[index] = P[index]/P[index].sum()
fig = plt.figure(figsize=(10,6))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,Q,s=1,alpha=0.5)
###
```

Out[197]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x24b0a14f550>



In [198]:
```
### REPLACE BY YOUR CODE
Qx = Q.sum(axis=0)
plt.figure(figsize=(7,4))
plt.plot(R,Qx,'o',ms=3)
plt.xlabel('x')
plt.ylabel('Q(x)')
plt.grid(True)
###
```



1.4 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

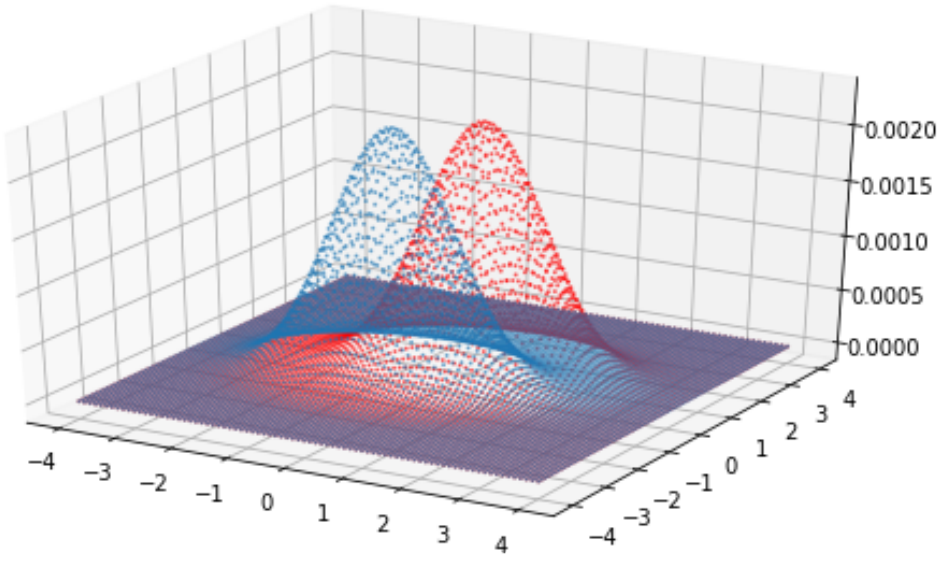
$$\Sigma = \begin{pmatrix} 1.0 & 0 \\ 0 & 0.5 \end{pmatrix}.$$

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.**

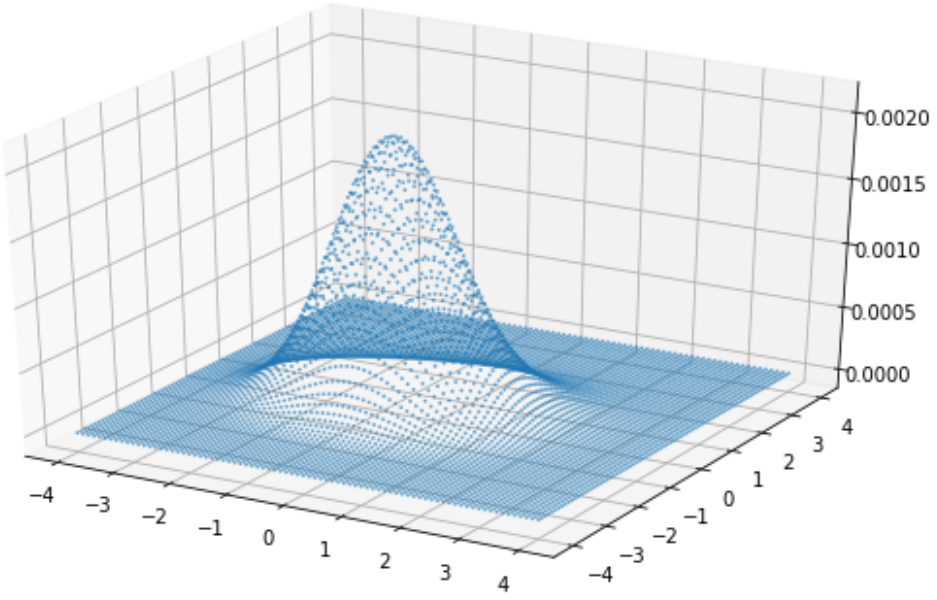
In [199]:
```
### REPLACE BY YOUR CODE
mu_1 = numpy.array([-.5,-.5])
mu_2 = numpy.array([.5,.5])
Cov = numpy.array([[1.0,0],[0,.5]])
Cov_inv = numpy.linalg.inv(Cov)
z = 2*numpy.pi*numpy.linalg.det(Cov)**.5
V = numpy.array(list(zip(X.flatten(),Y.flatten())))

D_1 = numpy.exp((-1/2)*numpy.diag((V-mu_1).dot(Cov_inv).dot((V-mu_1).T)))/z
s1 = numpy.sum(D_1)
D_1 = D_1/s1
fig = plt.figure(figsize=(9,5))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,D_1,s=1,alpha=0.5)

D_2 = numpy.exp((-1/2)*numpy.diag((V-mu_2).dot(Cov_inv).dot((V-mu_2).T)))/z
s2 = numpy.sum(D_2)
D_2 = D_2/s2
ax.scatter(X,Y,D_2,s=1,alpha=0.5,color='red')
###
```

Out[199]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x24b0b523f28>
```



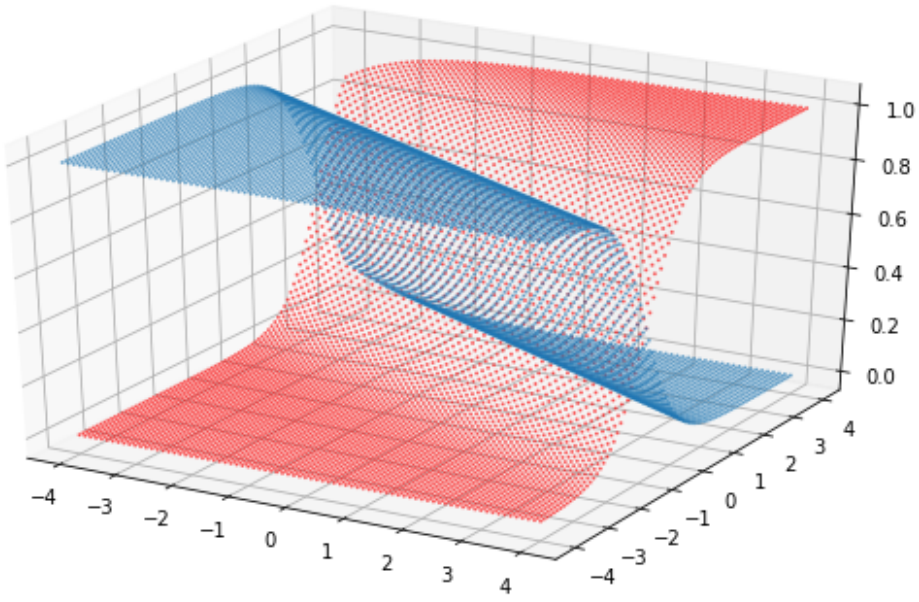
In [200]:
```
### REPLACE BY YOUR CODE
P_w1 = 0.9
P_w2 = 0.1
P_x = D_1*P_w1 + D_2*P_w2
fig = plt.figure(figsize=(10,6))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P_x,s=1,alpha=0.5)
###
```

Out[200]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x24b0b4ff6a0>



In [201]:
```
### REPLACE BY YOUR CODE
P_w1_x = D_1 * P_w1/P_x
P_w2_x = D_2 * P_w2/P_x

fig = plt.figure(figsize=(10,6))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P_w1_x,s=1,alpha=0.5)
ax.scatter(X,Y,P_w2_x,s=1,alpha=0.5,color='red')

error1 = numpy.where((P_w1_x >= P_w2_x))
error2 = numpy.where((P_w1_x < P_w2_x))
P_error = numpy.sum(P_w2_x[error1]*P_x[error1]) + numpy.sum(P_w1_x[error2]*P_x[error2]
print('Bayes Error Rate:','%.3f'%P_error)
###
```

```
Bayes Error Rate: 0.080
```



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

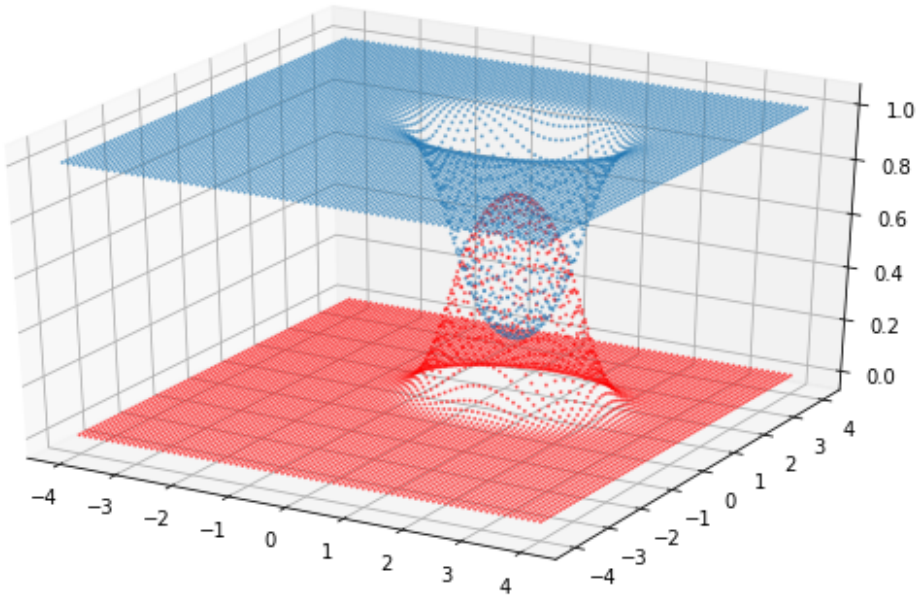
Suppose that the data generating distribution for the second class changes to produce samples much closer to the mean. This variance reduction for the second class is implemented by keeping the first covariance the same (i.e. $\Sigma_1 = \Sigma$) and dividing the second covariance matrix by 4 (i.e. $\Sigma_2 = \Sigma/4$). 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 [202]:
```
### REPLACE BY YOUR CODE
V = numpy.array(list(zip(X.flatten(),Y.flatten())))
```

```
Cov_1 = numpy.array([[1.0,0],[0,.5]])
Cov_1_inv = numpy.linalg.inv(Cov_1)
z_1 = 2*numpy.pi*numpy.linalg.det(Cov_1)**.5
D_1_new = numpy.exp((-1/2)*numpy.diag((V-mu_1).dot(Cov_1_inv).dot((V-mu_1).T)))/z_1
s1_new = D_1_new.sum()
D_1_new = D_1_new/s1_new
Cov_2 = Cov_1/4
Cov_2_inv = numpy.linalg.inv(Cov_2)
z_2 = 2*numpy.pi*numpy.linalg.det(Cov_2)**.5
D_2_new = numpy.exp((-1/2)*numpy.diag((V-mu_2).dot(Cov_2_inv).dot((V-mu_2).T)))/z_2
s2_new = D_2_new.sum()
D_2_new = D_2_new/s2_new
P_x_new = D_1_new * P_w1 + D_2_new * P_w2
P_w1_x_new = D_1_new * P_w1/P_x_new
P_w2_x_new = D_2_new * P_w2/P_x_new

fig = plt.figure(figsize=(10,6))
ax = plt.axes(projection='3d')
ax.scatter(X,Y,P_w1_x_new,s=1,alpha=0.5)
ax.scatter(X,Y,P_w2_x_new,s=1,alpha=0.5,color='red')

error1_new = numpy.where((P_w1_x_new >= P_w2_x_new))
error2_new = numpy.where((P_w1_x_new < P_w2_x_new))
P_error_new = numpy.sum(P_w2_x_new[error1_new]*P_x_new[error1_new]) + numpy.sum(P_w1_x
print('Bayes Error Rate:','%.3f'%P_error_new)
###
```

```
Bayes Error Rate: 0.073
```



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.**

Even if class 2 concentrate in the small region close to its center (0.5,0.5) with a smaller convariance, the prior probability of class 1 is much bigger than class 2. And the distribution of class 1 is seperated wider as class 2, so inside the input space the decision will refer to class 1 rather class 2.