# assignment1

October 4, 2019

This is the first assignment for the course Introduction to Machine Learning (COMP0088). The goal of this assignment is to learn how to visualize functions using the Python language, along with the numpy and matplotlib packages. This assignment takes the form of a Jupyter Notebook, a convenient way to combine formatted text, interactive coding and plotting.

## 1 Function visualization in Python (40%)

Consider the following function:

$$f(x_1, x_2; w) = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_1^2 + w_4 * x_2^2 + w_5 * x_1 * x_2$$

where

$$w = [-1, 0, 0, 4, 2, 0]'$$

To visualize this function in the domain $x \in [1, 1] \times [1, 1]$ we will sample its values with a step of 0.01 units, which gives us a $201 \times 201$ matrix with values:

$$F[i, j] = f\left(\frac{i}{100} - 1, 1 - \frac{j}{100}; w\right)$$

- Question 1 (0%) Try to understand how the visualization function works and in particular how the matplotlib functions `imshow`, `plot_surface` and `contour` are used. Run the two examples given. Do the next questions using these examples as a guideline.
- Question 2 (20%) Compute this matrix using a nested for-loop iterating over the values in `X1` and `X2`.
- Question 3 (20%) Compute this matrix again, this time applying the arithmetic operations directly to the numpy arrays `X1` and `X2`. Notice that this is both more efficient and requires fewer lines of code.
- Question 4 (0%) For classification we care a lot about the zero set of the function, where the function flips from being positive to being negative. Go back to the cells of questions 2 and 3 and tweak the parameters of the visualization function to also plot the zero set of $f$.

## 2 Visualizing a classifier's decision boundary (60%)

Building on the previous section, consider now the following basic problem discussed in class: you have a two-class classification problem. The prior probabilities of the two classes are $\pi_0 = 0.3$ and $\pi_1 = 1 - \pi_0 = 0.7$, while for both classes the class-conditional probability density function is of the following Gaussian form:

$$P(X = x | C = c) = \frac{1}{2\pi\sigma_1^c\sigma_2^c} \exp\left(-\frac{(x_1 - \mu_1^c)^2}{2(\sigma_1^c)^2} - \frac{(x_2 - \mu_2^c)^2}{2(\sigma_2^c)^2}\right)$$

where the parameters for the two classes are considered to be the following:

$$C = 0: \quad \mu_1^0 = -0.2, \quad \mu_2^0 = 0.8, \quad \sigma_1^0 = 1.0, \quad \sigma_2^0 = 0.2$$
$$C = 1: \quad \mu_1^1 = 0.7, \quad \mu_2^1 = -0.8, \quad \sigma_1^1 = 0.5, \quad \sigma_2^1 = 0.6$$

This expression is provided in matrix form in the course (see Week 2: Logistic Regression, slide 63).

- Question 5 (20%) Define a function that calculates this class-conditional probability density function, and adapt your code from the previous section to compute the isocontours of the distributions of the two classes given, at the values of 0.1 and 0.2.
- Question 6 (20%) Plot the decision boundary for the posterior-based classifier using Bayes' rule:

$$P(C = 1 | X = x) = \frac{P(X = x | C = 1)P(C = 1)}{\sum_{c \in \{0,1\}} P(X = x | C = c)P(C = c)}$$

First, define a function computing the posterior probability for class 1 (reusing code from the previous question). Then, as in the previous examples, plot the isocontour of this function at $P(C = 1 | X = x) = 0.5$, which is where the decision outcome changes.

- Question 7 (20%) Consider now that $\pi_0 = 0.1$ and repeat. What do you observe?

## 3 Code

### 3.1 Imports

```
[1]: import os
     import time

     import matplotlib.pyplot as plt
     import numpy as np
     import scipy.io as spio
     from mpl_toolkits.mplot3d import Axes3D

     %matplotlib inline
```

### 3.2 Visualization function

```
[2]: def visualization_func(grid_x, grid_y, function_values, contour_values=[0.5]):
         # Create the matplotlib figure
         fig = plt.figure(figsize=plt.figaspect(0.3))
```

```python
    # Add the first subplot, known as an 'Axes' object in matplotlib
    ax = fig.add_subplot(1, 3, 1)
    # Produce an image plot of the values. By default, values are interpolated
    # between points to produce a smooth visual.
    ax.imshow(function_values, extent=[-1, 1, -1, 1], origin='lower')
    ax.set_title('Image plot of function values, clipped between [-1,1]')

    ax = fig.add_subplot(1, 3, 2, projection='3d')
    # Produce another plot of these values, this time as a surface in 3D space.
    surf = ax.plot_surface(grid_x, grid_y, function_values, rstride=1, cstride=1,
                           linewidth=0, antialiased=False)
    ax.set_zlim3d(-1, 1)
    ax.set_title('Mesh plot of function values')

    ax = fig.add_subplot(1, 3, 3)
    # Plot some contour lines of our function. Again, these are automatically
    # interpolated from the values.
    CS = ax.contour(grid_x, grid_y, function_values, contour_values,
                    cmap=plt.cm.winter)
    ax.clabel(CS)
    ax.set_title('Isocontours of function')

    # Finally, show the figure.
    plt.show()
```

### 3.2.1 Sample points using `meshgrid`

```python
[3]: num_of_points = 201
    # Linearly sample the [-1,1] interval with the given number of points.
    # The retstep option returns the step as well as the values.
    x1, step = np.linspace(-1.0, 1.0, num=num_of_points, retstep=True)
    # Print the step size to confirm it is correct
    print('Step size is : {}'.format(step))
    x2 = np.linspace(-1.0, 1.0, num=num_of_points)

    # Create a rectangular grid out of the x1 and x2 values.
    # Each point of this grid is a 2D point with coordinates (X1[i, j], X2[i, j]).
    # See https://stackoverflow.com/a/36014586 for an illustration.
    X1, X2 = np.meshgrid(x1, x2, sparse=False)
```
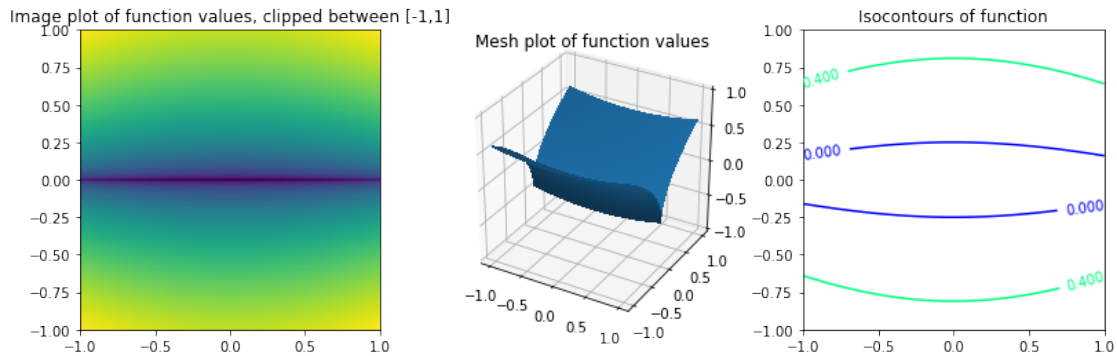
```
Step size is : 0.01
```

### 3.2.2 Example 1

Generate values for $f(x_1, x_2) = 0.1 * x_1^2 + \sqrt{|x_2|} - .5$

```
[4]: function_values = 0.1 * X1**2 + np.sqrt(np.abs(X2)) - 0.5

     visualization_func(X1, X2, function_values, contour_values=[0.0, 0.4])
```
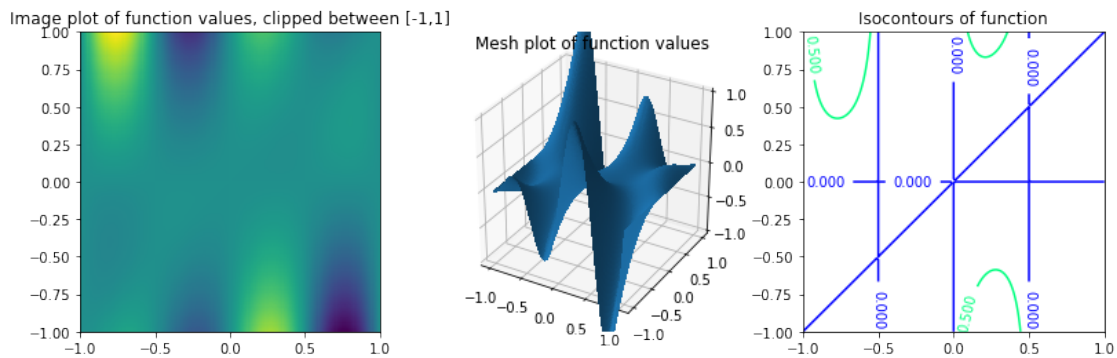


### 3.2.3   Example 2

Generate values for $f(x_1, x_2) = \sin(2 * \pi * x_1) * (x_2^2 - x_1 * x_2)$

```
[5]: function_values = np.sin(2*np.pi*X1) * (X2**2 - X1*X2)

     visualization_func(X1, X2, function_values, contour_values=[0.0, 0.5])
```



## 3.3   Matrix computation

### 3.3.1   Compute function values using a nested for-loop

```
[6]: w = [-1, 0, 0, 4, 2, 0]

     rows, cols = X1.shape
```

```
function_values = np.zeros((rows, cols))

t = time.process_time()

## TODO (Question 2)

## /TODO

elapsed = time.process_time() - t
print(elapsed, ' seconds have passed using a nested for-loop')

## TODO (Question 4: tweak parameters to plot the zero set)

## /TODO
```

0.11616937300000174  seconds have passed using a nested for-loop
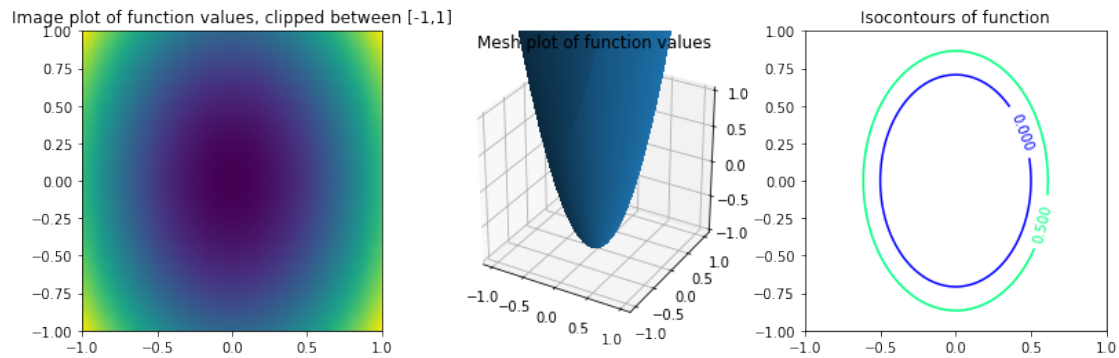


### 3.3.2  Compute using numpy operations

```
[7]: t = time.process_time()

## TODO (Question 3)

## /TODO

elapsed = time.process_time() - t
print(elapsed, ' seconds have passed using numpy operations')

## TODO (Question 4: tweak parameters to plot the zero set)

## /TODO
```
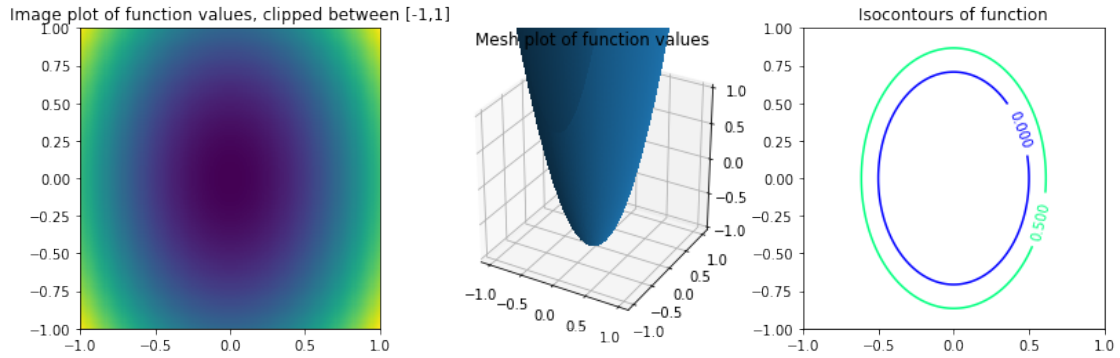
0.00168910000000011  seconds have passed using numpy operations

Image plot of function values, clipped between [-1,1]  Mesh plot of function values  Isocontours of function

## 3.4 Decision boundary of a classifier

### 3.4.1 Probability function

```python
[8]: def probability_func(X1, X2, m_1, m_2, sigma_1, sigma_2):
         ## TODO (Question 5)

         ## /TODO

         return output
```
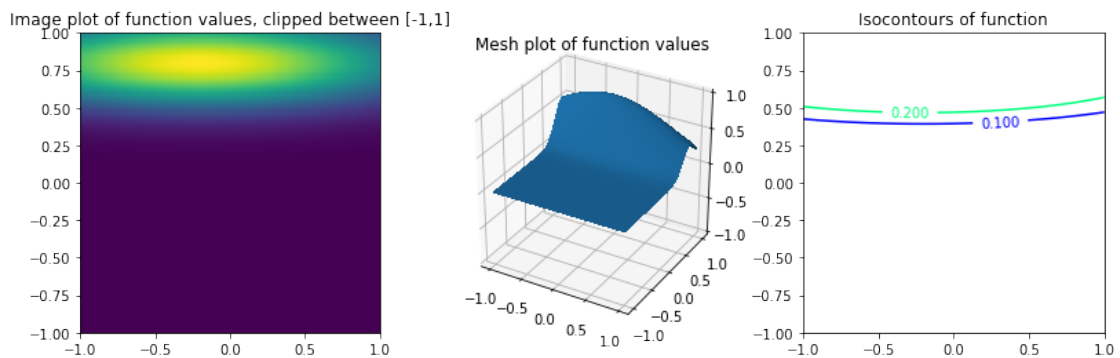
**Visualize the distribution of the first class**

```python
[9]: m0_1 = -0.2
     m0_2 = 0.8
     sigma0_1 = 1
     sigma0_2 = 0.2

     ## TODO (Question 5)

     ## /TODO
```



Image plot of function values, clipped between [-1,1]  Mesh plot of function values  Isocontours of function
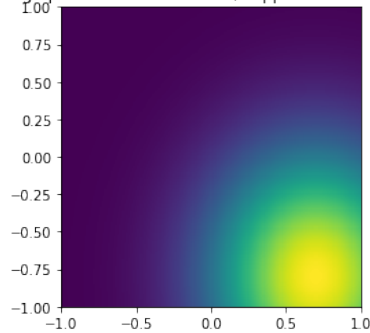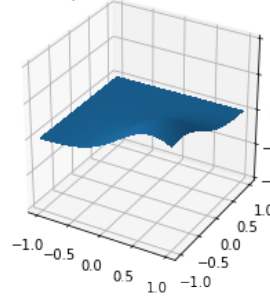
**Visualize the distribution of the second class**

```
[10]: m1_1 = 0.7
      m1_2 = -0.8
      sigma1_1 = 0.5
      sigma1_2 = 0.6

      ## TODO (Question 5)

      ## /TODO
```
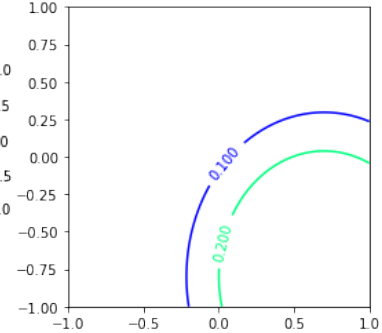


Image plot of function values, clipped between [-1,1] — Mesh plot of function values — Isocontours of function

### 3.4.2  Decision boundary function

```
[11]: def decision_boundary_function(X1, X2, prior0):

          ## TODO (Question 6)

          # /TODO

          return posterior1
```
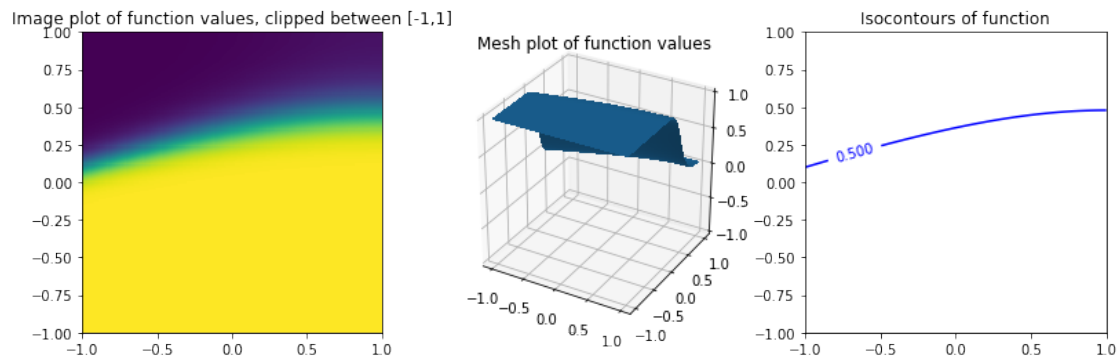
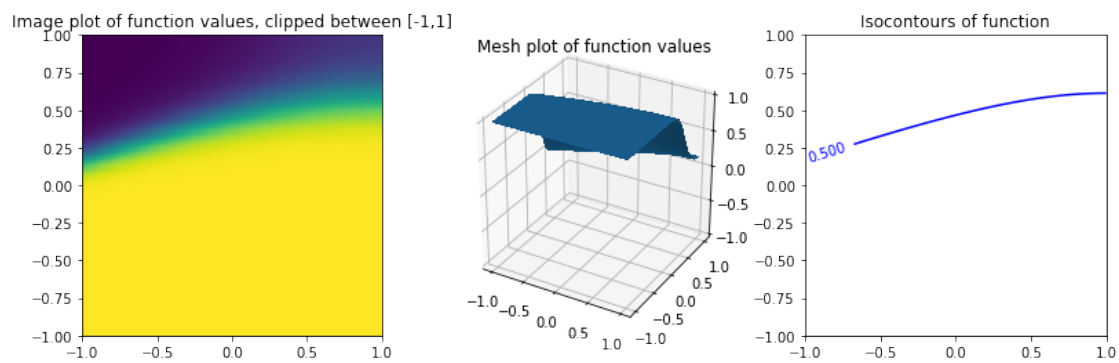**Visualize the decision boundary for $\pi_0 = 0.3$**

```
[12]: ## TODO (Question 6)

      ## /TODO
```

**Visualize the decision boundary for $\pi_0 = 0.1$**

```
[13]:  ## TODO (Question 7)

       ## /TODO
```



**What do you observe?**

```
[1]:  ## TODO (Question 7)

      ## /TODO
```