Visualization

October 9, 2017

1 Function Visualization in Python (0.4)

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 value with a step of .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 (.2) Compute this matrix using a nested for-loop.
- Question 3 (.2) Compute this matrix more efficiently, using numpy's vectorized operations.
- 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. Tweak the parameters of the visualization function so that you also plot the zero set of f using numpy's *contour*.

2 Visualizing a classifier's decision boundary (0.6)

Building on the previous assignment, 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:

$$\mu_1^0 = -.2, \mu_2^0 = .8, \sigma_1^0 = 1, \sigma_2^0 = .2 \mu_1^1 = .7, \mu_2^1 = -.8, \sigma_1^1 = .5, \sigma_2^1 = .6$$

- Question 5 (.2) Define a function that calculates the class probability, and adapt your code from the previous assignment to compute the isocontours of the distributions of the two classes given, at the values of 0.1 and 0.2.
- Question 6 (.2) 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 (.2) Consider now that $\pi_0 = .1$ and repeat. What do you observe?

3 Code

3.1 Imports

```
In [1]: import os
        import numpy as np
        import matplotlib.pyplot as plt
        import scipy.io as spio
        from numpy import pi, sin, abs, sqrt, exp
        from mpl_toolkits.mplot3d import Axes3D
        import time

        %matplotlib inline
```

3.2 Visualization function

```
In [2]: def visualization_func(grid_x, grid_y, function_values, contour_values=[0.5]):
            fig = plt.figure(figsize=plt.figaspect(0.3))
            ax = fig.add_subplot(1, 3, 1)
            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')
            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)
            CS = ax.contour(grid_x, grid_y, function_values, contour_values,
                            cmap=plt.cm.winter)
            ax.clabel(CS)
            ax.set_title('Isocontours of function')

            plt.show()
```

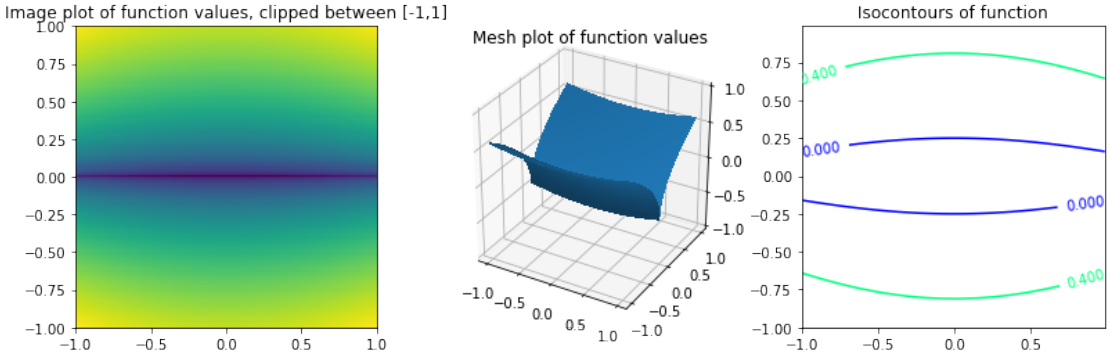
3.2.1 Sample points using *meshgrid*

```
In [3]: x1 = np.arange(-1, 1, 0.01)
        x2 = np.arange(-1, 1, 0.01)

        X1, X2 = np.meshgrid(x1, x2, sparse=False)
```

3.2.2 Example 1

Generate values for $f(x_1, x_2) = 0.1 * x\_1^2 + \sqrt{|x_2|} - .5$

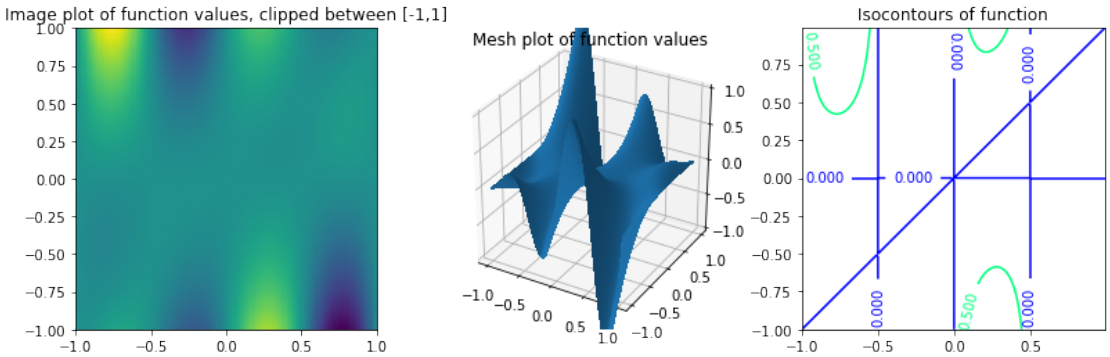
```
In [4]: function_values = 0.1*X1**2 + np.sqrt(np.abs(X2)) - 0.5

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



3.2.3 Example 2

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

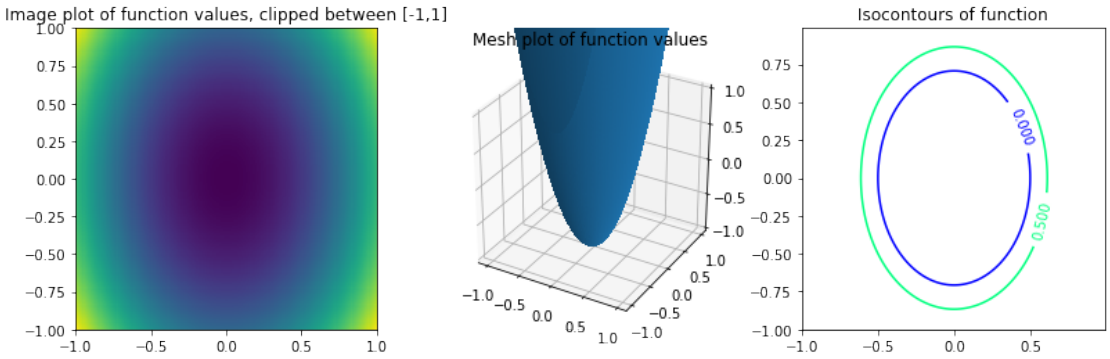
```
In [5]: function_values = sin(2*pi*X1)*(X2**2 - X1*X2)

        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

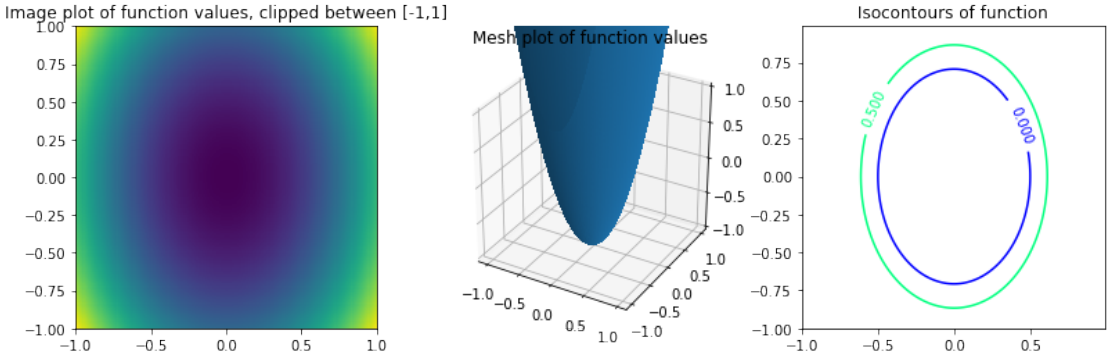
```
In [6]: w = [-1, 0, 0, 4, 2, 0]

        rows, cols = X1.shape

        function_values = np.zeros([rows, cols], dtype=np.float32)

        t = time.time()

        ## TODO (Question 2)

        ## /TODO

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

        ## TODO (Question 4: tweak parameters to plot the zero set)
        visualization_func(X1, X2, function_values)
        ## /TODO
```



3.3.2 Compute using numpy's vectorized operations

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

        ## TODO (Question 3)

        ## /TODO

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

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

```
    visualization_func(X1, X2, function_values)
    ## /TODO
```



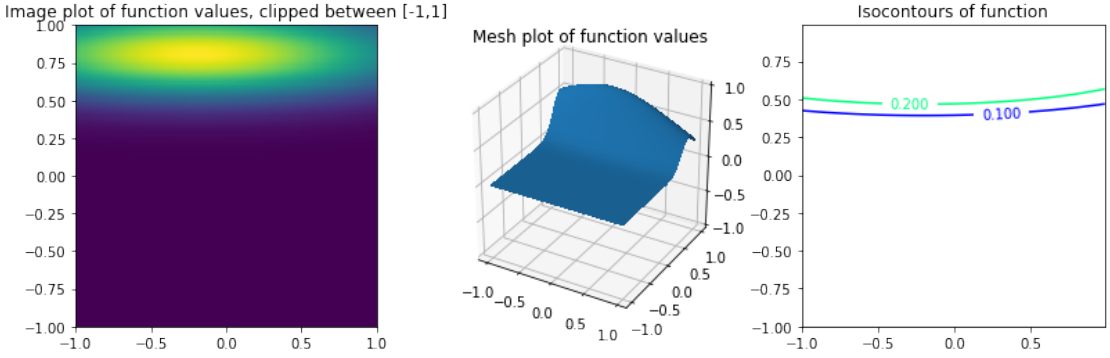
3.4 Decision boundary of a classifier

3.4.1 Probability function

```
In [8]: def probability_func(X1, X2, m1, m2, sigma1, sigma2):
            ## TODO (Question 5)

            ## /TODO

            return output
```

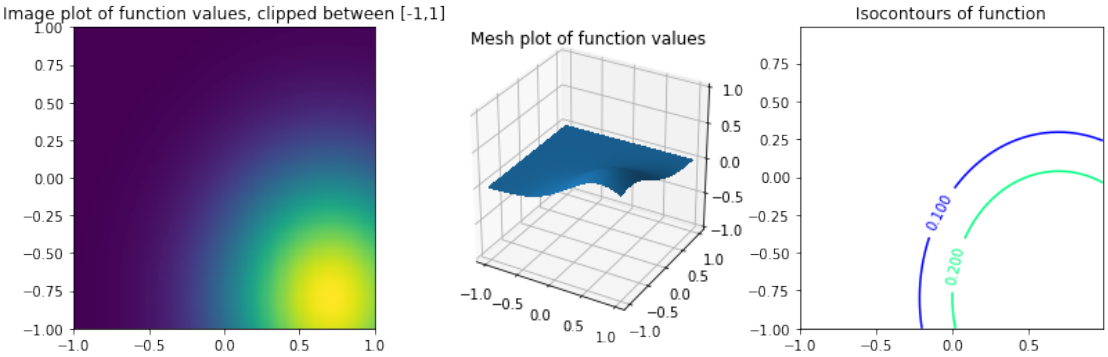
3.4.2 Visualize the distribution of the first class

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

        ## TODO (Question 5)

        ## /TODO
```



3.4.3 Visualize the distribution of the second class

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

         ## TODO (Question 5)

         ## /TODO
```

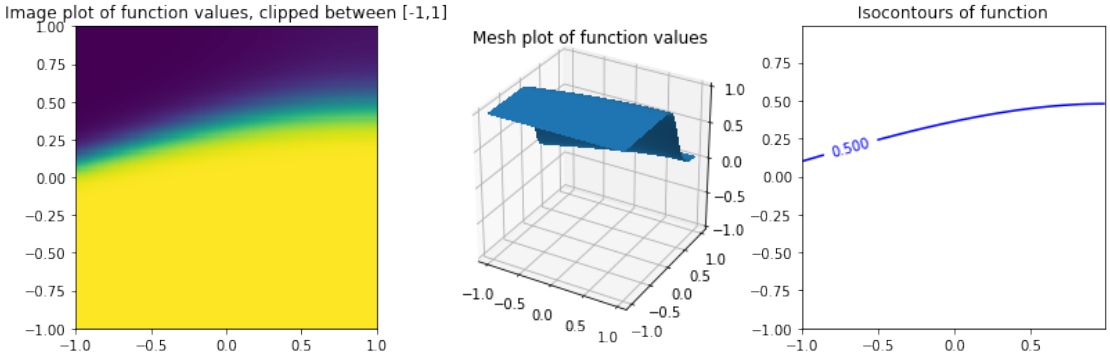


3.5 Decision boundary function

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

             ## TODO (Question 6)

             # /TODO

             return posterior1
```

3.5.1 Visualize the decision boundary for $\pi_0 = 0.3$

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

         ## /TODO
```



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

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

         ## /TODO
```