labsol3

February 14, 2022

1 Machine Learning - Laboratory 3

[1]:
```
import getopt
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from matplotlib import cm
from pdffuns import *
import pickle
```

[2]:
```
def labsol3(met='ML', discr='pxw', prm = []):

    # Initialise values:
    # - axes, x1 and x2
    x1 = np.arange(-6.25,6.26,0.5).reshape(-1,1)
    x2 = np.arange(-6.25,6.26,0.5).reshape(-1,1)

    # Get coordinates grid
    X1, X2 = np.meshgrid(x1, x2)

    # Pack everything
    Xgrid = np.dstack((X1, X2))

    # Load data from pickle files
    pfile = 'lab3.p'
    with open(pfile, "rb") as fp:
        X=pickle.load(fp)

    # Number of classes
    M = len(X)

    # Feature dimension
    l = len(X[0])

    # Estimate prior probabilities, Pwi[k].
    N, Pw = [], []
```

```
    # Number of feature vectors: N[i] --> Feature vectors of class i
    for i in range(M):
        N.append(len(X[i][0]))

    # Determine Pwi
    for i in range(M):
        Pw.append(N[i]/sum(N))


    # Initialise method specific parameters
    # - on condition of met
    # - Maximum likelihood: my, Sgm (empty)
    # - Parzen window: h1 from prm
    # - kn nearest neighbor: knn from prm
    if met == 'ML':
        my = np.zeros(shape=(M, l), dtype=float)
        Sgm = np.zeros(shape=(M, l, l), dtype=float)
    if met == 'knn':
        my = None
        Sgm = None
        kn = prm[0]
    if met == 'PZ':
        my = None
        Sgm = None
        h1 = prm[0]
    # - parameters, my[i] and Sgm[i], i = 0,...,M-1
    # - prior probabilities, Pw[i], i = 0,...,M-1

    # Determine class specific probability density functions, pxw[i], i = 0,...
↪,M-1
    # - initialise pxw as empty list
    pxw = np.zeros(shape=(M, np.shape(Xgrid)[0], np.shape(Xgrid)[1]))

    g = np.zeros(shape=pxw.shape)
    # - initialise total density function, p as zero
    p = 0
    # - iterate over classes, k = 0,...,M-1
    for i in range(M):

        # - on condition of met: Maximum likelihood:
        if met == 'ML':
            # - estimate parameters my[k], Sgm[k]
            for j in range(l):
                my[i][j] = np.mean(X[i][j])

            Sgm[i] = np.cov(X[i])
```

```
            # - determine pxw[k] by using norm2D
            # Reshape 'my' to pass it as a column vector
            pxw[i, :, :] = norm2D(my[i].reshape(-1, 1), Sgm[i], Xgrid)

        # - on condition of met: Parzen window:
            # - initialise pxw[k]
            # - determine hn
            # - iterate over samples in X[k], i = 1,...,N[k]
                # - let x be feature vector number i in X[k]
                # - use norm2D to determine pN
                # - update pxw[k] by adding pN
            # - update pxw by dividing by N[k]
        # - on condition of met: kn-nearest neighbor:
        if met == "knn":
            # - use knn2D to determine pxw[k] from X[k]
            pxw[i, :, :] = knn2D(X[i], Xgrid, kn)
        if met == "PZ":
            hn = h1/np.sqrt(N[i])
            hnI = hn**2 * np.eye(l)

            # iterate over all feature vectors in class i
            for j in range(0, N[i]):
                # feature vector j of class i
                xk = X[i][:,j].reshape(l,1)
                # sum up the probabilities of each of the N[i] distributions
                # Note that there is one distribution for each datapoint!
                pxw[i, :, :] = pxw[i, :, :] + norm2D(xk, hnI, Xgrid)
                # divide by number of feature vectors in class i
                pxw[i, :, :] /= N[i]

        # - update p
        p += Pw[i] * pxw[i]

    # Determine discriminant functions, g[k], k = 0,...,M-1
    for i in range(M):
        g[i] = (Pw[i] * pxw[i])

    return x1, x2, my, Sgm, g/p, g
```

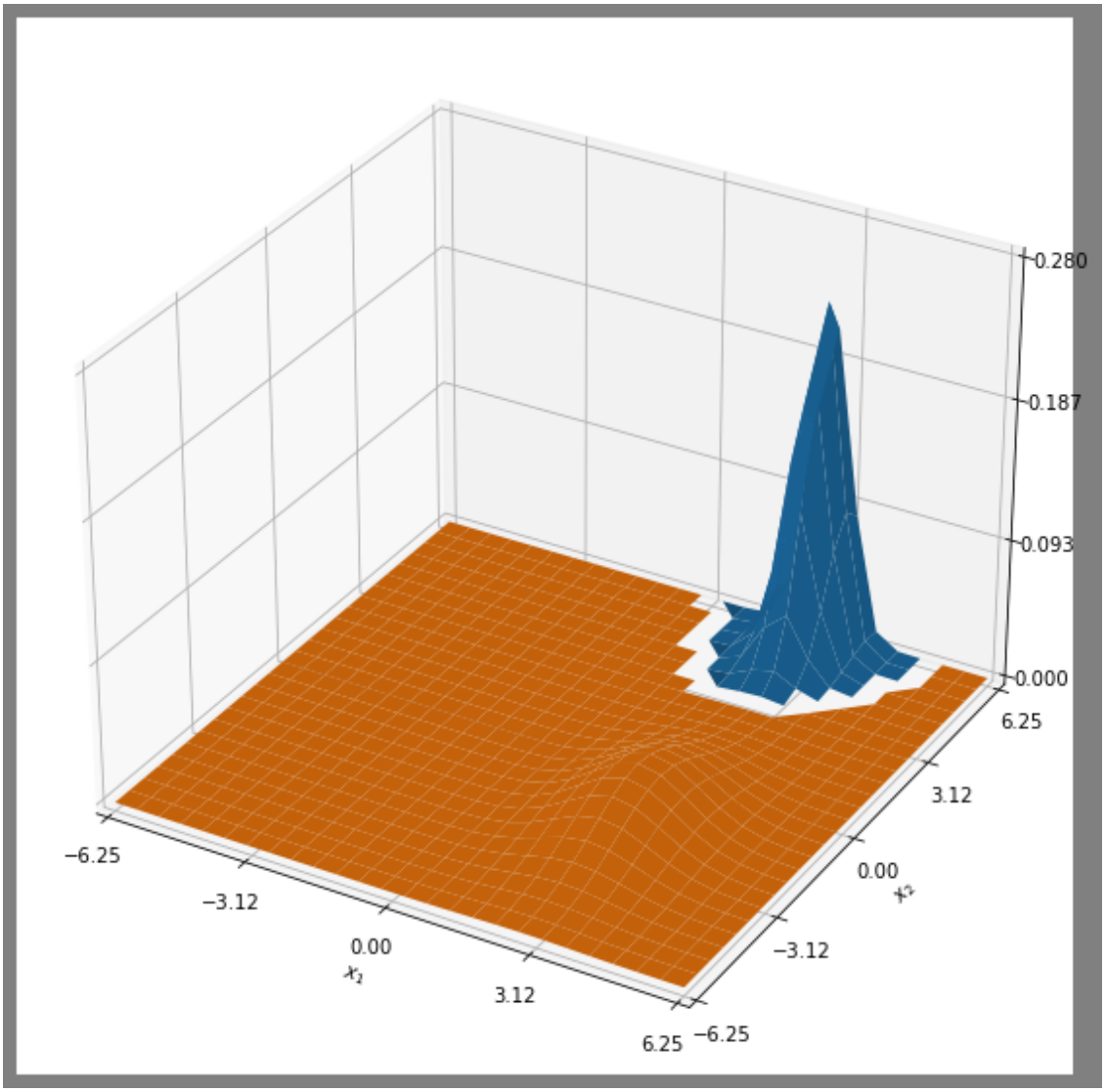
1.1 Sections a), b), c) and d)

[3]:
```
# Call the function to get the data
x1, x2, my, Sgm, posterior, df= labsol3()

# Print estimated parameters
```

```
print(f"Estimated mean vectors: \n{my[0].reshape(-1, 1)} \n {my[1].reshape(-1,␣
↪1)}")
print(f"Estimated covariance matrices: \n{Sgm}")

# Plot the discriminant functions
classplot(df, x1, x2, 1, gsv={'gsv': 1, 'figstr': 'pdf'})
```

```
Estimated mean vectors:
[[2.65 ]
 [5.825]]
 [[ 2.8       ]
 [-1.76666667]]
Estimated covariance matrices:
[[[ 0.19666667  0.005     ]
  [ 0.005       0.73583333]]

 [[ 1.11       -0.01      ]
  [-0.01        2.92333333]]]
```

```
C:\Users\aeste\OneDrive - UNIVERSIDAD DE HUELVA\Universidad\4º Carrera
(Stavanger)\Spring Semester\ML\ML\pdffuns.py:100: MatplotlibDeprecationWarning:
Calling gca() with keyword arguments was deprecated in Matplotlib 3.4. Starting
two minor releases later, gca() will take no keyword arguments. The gca()
function should only be used to get the current axes, or if no axes exist,
create new axes with default keyword arguments. To create a new axes with non-
default arguments, use plt.axes() or plt.subplot().
  ax = fig.gca(projection='3d')
C:\Users\aeste\OneDrive - UNIVERSIDAD DE HUELVA\Universidad\4º Carrera
(Stavanger)\Spring Semester\ML\ML\pdffuns.py:112: UserWarning: Z contains NaN
values. This may result in rendering artifacts.
  obj = ax.plot_surface(X1, X2, G, facecolor=col[i])
```

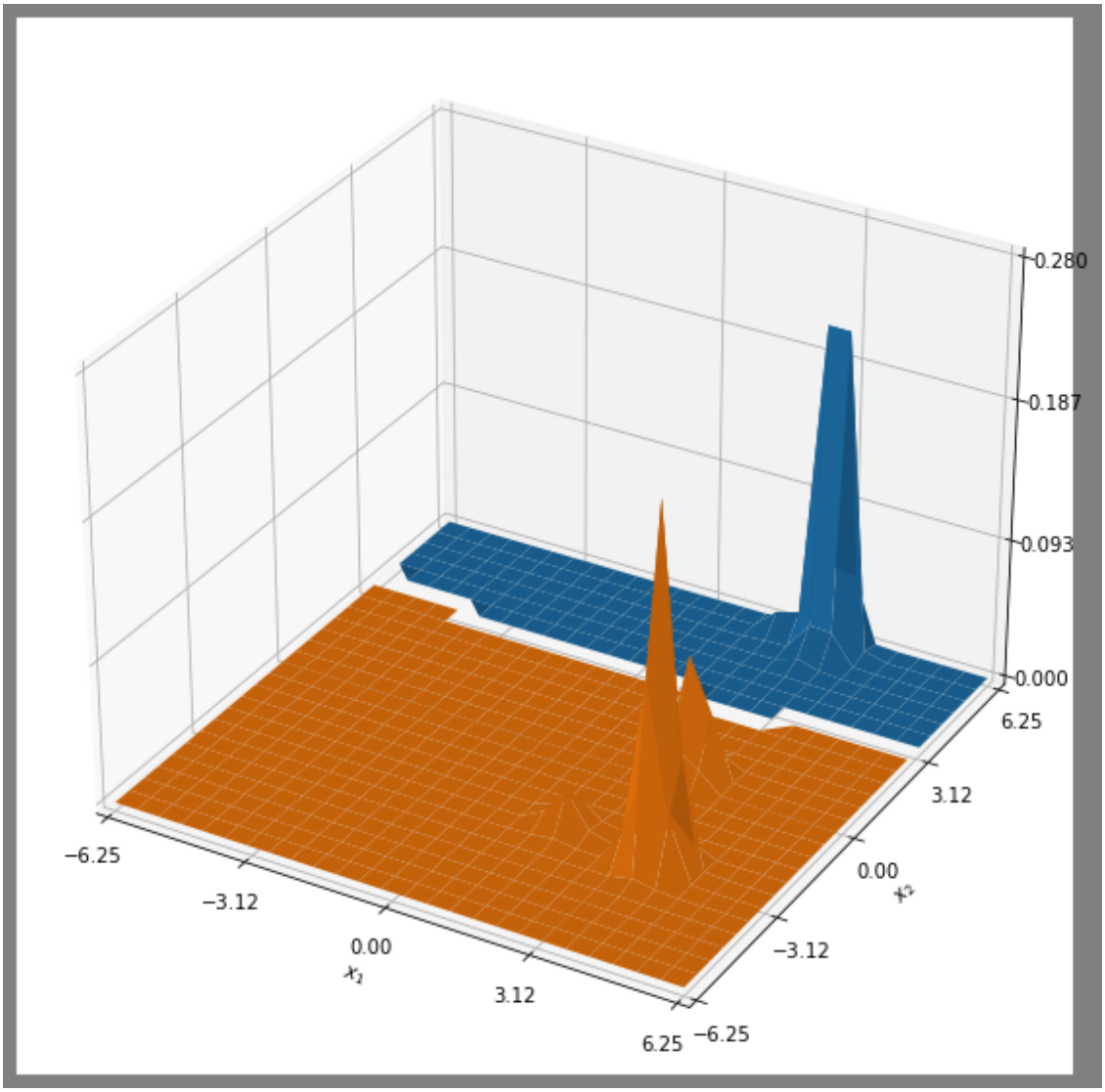


1.2 Section e)

If we compare this discriminant functions to the ones that we got in **labsol2.ipynb**, we can notice they are quite similar. Nevertheless, we can also see that the decision border is less smoothed, this could be directly caused by the few amount of data we have to estimate the parameters μ and Σ.

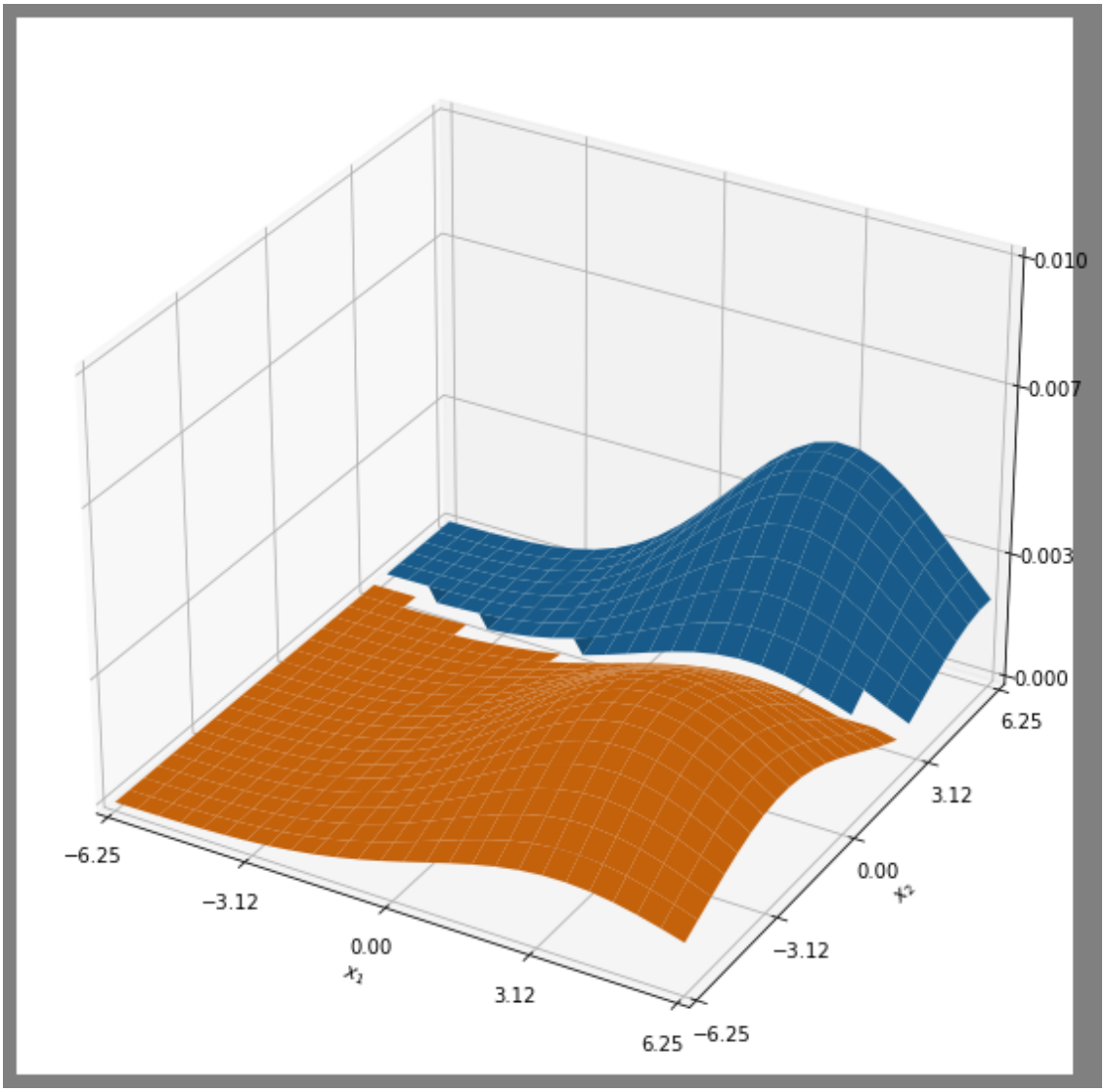
1.3 Section f)

[4]:
```
x1, x2, my, Sgm, posterior, df = labsol3('PZ', 'pxw', [0.5])

classplot(df, x1, x2, 1, gsv={'gsv': 1, 'figstr': 'pdf'})
```



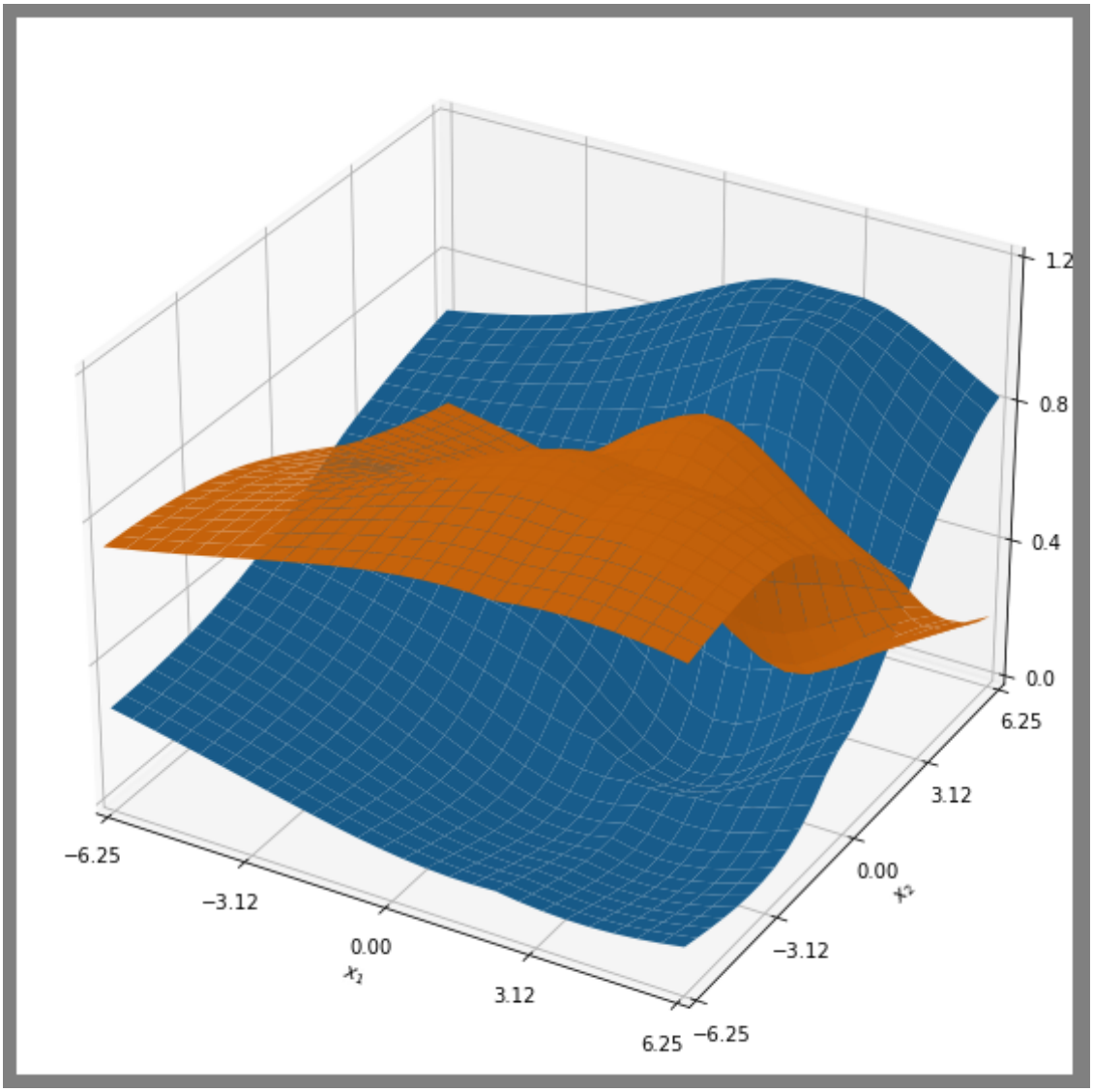
1.4 Section g)

[5]:
```
# Call the function to get the data
x1, x2, my, Sgm, posterior, df = labsol3('PZ', 'pxw', [5.])

classplot(df, x1, x2, 1, gsv={'gsv': 1, 'figstr': 'pdf'})
```



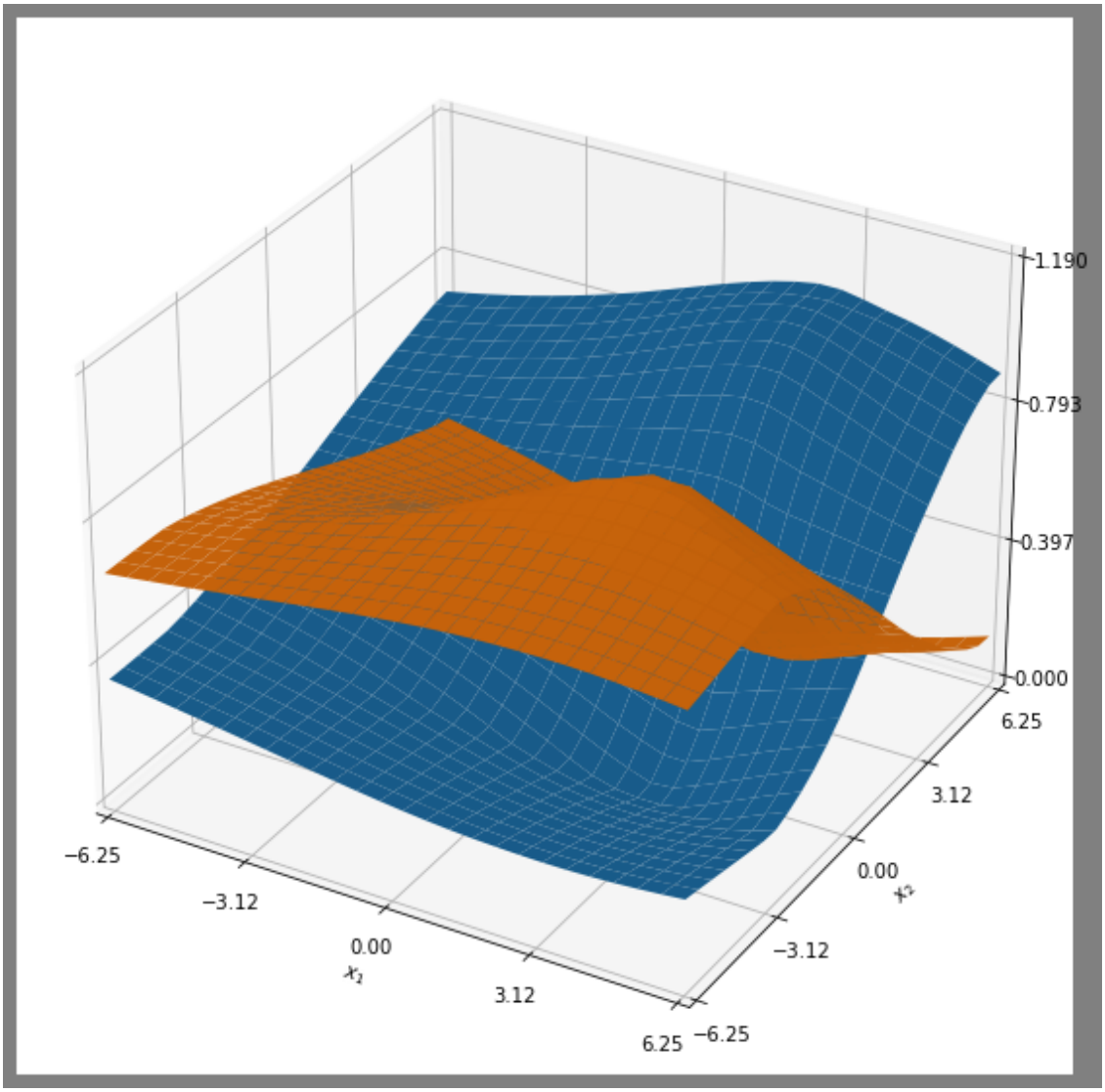
1.5 Section h)

[6]:
```
# Call the function to get the data
x1, x2, my, Sgm, posterior, df= labsol3('knn', 'pxw', [1])

# Plot the discriminant functions
classplot(posterior, x1, x2, 0, gsv={'gsv': 1, 'figstr': 'pdf'})
```



1.6 Section i)

[7]:
```
# Call the function to get the data
x1, x2, my, Sgm, posterior, df = labsol3('knn', 'pxw', [3])

# Plot the discriminant functions
classplot(posterior, x1, x2, 0, gsv={'gsv': 1, 'figstr': 'pdf'})
```



1.7 Section j)

[8]:
```
# Call the function to get the data
x1, x2, my, Sgm, posterior, df = labsol3('knn', 'pxw', [5])

# Plot the discriminant functions
classplot(posterior, x1, x2, 0, gsv={'gsv': 1, 'figstr': 'pdf'})
```

```
C:\Users\aeste\AppData\Local\Temp/ipykernel_8700/3380010590.py:111:
RuntimeWarning: invalid value encountered in true_divide
  return x1, x2, my, Sgm, g/p, g
C:\Users\aeste\anaconda3\lib\site-packages\mpl_toolkits\mplot3d\art3d.py:34:
RuntimeWarning: invalid value encountered in double_scalars
  a = (a + 180) % 180
```

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values

kn can't be bigger than the size of the dataset
kn can't be bigger than the size of the dataset

posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8700/947789179.py in <module>
      3
      4 # Plot the discriminant functions
----> 5 classplot(posterior, x1, x2, 0, gsv={'gsv': 1, 'figstr': 'pdf'})

~\OneDrive - UNIVERSIDAD DE HUELVA\Universidad\4º Carrera (Stavanger)\Spring␣
 ↪Semester\ML\ML\pdffuns.py in classplot(g, x1, x2, gnan, discr, gsv)
    121     ax.set(xlabel='$x_1$', ylabel='$x_2$', zlabel=zlb)
    122     if gsv['gsv']:
--> 123         plt.savefig(gsv['figstr'] + discr + '.png')
    124
    125     plt.show()

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in savefig(*args, **kwargs)
    964 def savefig(*args, **kwargs):
    965     fig = gcf()
--> 966     res = fig.savefig(*args, **kwargs)
    967     fig.canvas.draw_idle()   # need this if 'transparent=True' to reset␣
 ↪colors
    968     return res

~\anaconda3\lib\site-packages\matplotlib\figure.py in savefig(self, fname,␣
 ↪transparent, **kwargs)
   3013                 patch.set_edgecolor('none')
   3014
-> 3015         self.canvas.print_figure(fname, **kwargs)
   3016
   3017         if transparent:
```

```
~\anaconda3\lib\site-packages\matplotlib\backend_bases.py in print_figure(self,␣
↪filename, dpi, facecolor, edgecolor, orientation, format, bbox_inches,␣
↪pad_inches, bbox_extra_artists, backend, **kwargs)
   2253                 # force the figure dpi to 72), so we need to set it␣
↪again here.
   2254                 with cbook._setattr_cm(self.figure, dpi=dpi):
-> 2255                     result = print_method(

   2256                         filename,
   2257                         facecolor=facecolor,

~\anaconda3\lib\site-packages\matplotlib\backend_bases.py in wrapper(*args,␣
↪**kwargs)
   1667             kwargs.pop(arg)
   1668
-> 1669         return func(*args, **kwargs)
   1670
   1671     return wrapper

~\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py in␣
↪print_png(self, filename_or_obj, metadata, pil_kwargs, *args)
    506             *metadata*, including the default 'Software' key.
    507         """
--> 508         FigureCanvasAgg.draw(self)
    509         mpl.image.imsave(
    510             filename_or_obj, self.buffer_rgba(), format="png",␣
↪origin="upper",

~\anaconda3\lib\site-packages\matplotlib\backends\backend_agg.py in draw(self)
    404              (self.toolbar._wait_cursor_for_draw_cm() if self.toolbar
    405               else nullcontext()):
--> 406             self.figure.draw(self.renderer)
    407             # A GUI class may be need to update a window using this␣
↪draw, so
    408             # don't forget to call the superclass.

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     72     @wraps(draw)
     73     def draw_wrapper(artist, renderer, *args, **kwargs):
---> 74         result = draw(artist, renderer, *args, **kwargs)
     75         if renderer._rasterizing:
     76             renderer.stop_rasterizing()

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
```

```
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\matplotlib\figure.py in draw(self, renderer)
   2788
   2789             self.patch.draw(renderer)
-> 2790             mimage._draw_list_compositing_images(
   2791                 renderer, self, artists, self.suppressComposite)
   2792

~\anaconda3\lib\site-packages\matplotlib\image.py in␣
↪_draw_list_compositing_images(renderer, parent, artists, suppress_composite)
    130     if not_composite or not has_images:
    131         for a in artists:
--> 132             a.draw(renderer)
    133     else:
    134         # Composite any adjacent images together

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\mpl_toolkits\mplot3d\axes3d.py in draw(self,␣
↪renderer)
    499                 # Then axes
    500                 for axis in self._get_axis_list():
--> 501                     axis.draw(renderer)
    502
    503             # Then rest

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\mpl_toolkits\mplot3d\axis3d.py in draw(self,␣
↪renderer)
    219         renderer.open_group('axis3d', gid=self.get_gid())
    220
```

```
--> 221         ticks = self._update_ticks()
    222
    223         info = self._axinfo

~\anaconda3\lib\site-packages\matplotlib\axis.py in _update_ticks(self)
   1027         """
   1028         major_locs = self.get_majorticklocs()
-> 1029         major_labels = self.major.formatter.format_ticks(major_locs)
   1030         major_ticks = self.get_major_ticks(len(major_locs))
   1031         self.major.formatter.set_locs(major_locs)

~\anaconda3\lib\site-packages\matplotlib\ticker.py in format_ticks(self, values
    260     def format_ticks(self, values):
    261         """Return the tick labels for all the ticks at once."""
--> 262         self.set_locs(values)
    263         return [self(value, i) for i, value in enumerate(values)]
    264

~\anaconda3\lib\site-packages\matplotlib\ticker.py in set_locs(self, locs)
    804                 self._compute_offset()
    805             self._set_order_of_magnitude()
--> 806             self._set_format()
    807
    808     def _compute_offset(self):

~\anaconda3\lib\site-packages\matplotlib\ticker.py in _set_format(self)
    900             # We needed the end points only for the loc_range␣
 ↪calculation.
    901             locs = locs[:-2]
--> 902         loc_range_oom = int(math.floor(math.log10(loc_range)))
    903         # first estimate:
    904         sigfigs = max(0, 3 - loc_range_oom)

ValueError: cannot convert float NaN to integer
```

```
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
posx and posy should be finite values
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\anaconda3\lib\site-packages\IPython\core\formatters.py in __call__(self, obj)
    339                 pass
    340             else:
--> 341                 return printer(obj)
    342             # Finally look for special method names
    343             method = get_real_method(obj, self.print_method)

~\anaconda3\lib\site-packages\IPython\core\pylabtools.py in print_figure(fig,␣
 ↪fmt, bbox_inches, base64, **kwargs)
    149         FigureCanvasBase(fig)
    150
--> 151     fig.canvas.print_figure(bytes_io, **kw)
    152     data = bytes_io.getvalue()
    153     if fmt == 'svg':

~\anaconda3\lib\site-packages\matplotlib\backend_bases.py in print_figure(self,␣
 ↪filename, dpi, facecolor, edgecolor, orientation, format, bbox_inches,␣
 ↪pad_inches, bbox_extra_artists, backend, **kwargs)
   2228                         else suppress())
   2229                 with ctx:
-> 2230                     self.figure.draw(renderer)
   2231
   2232             if bbox_inches:

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
 ↪renderer, *args, **kwargs)
     72     @wraps(draw)
     73     def draw_wrapper(artist, renderer, *args, **kwargs):
---> 74         result = draw(artist, renderer, *args, **kwargs)
     75         if renderer._rasterizing:
     76             renderer.stop_rasterizing()

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
 ↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\matplotlib\figure.py in draw(self, renderer)
   2788
   2789             self.patch.draw(renderer)
-> 2790             mimage._draw_list_compositing_images(
   2791                 renderer, self, artists, self.suppressComposite)
```

```
   2792

~\anaconda3\lib\site-packages\matplotlib\image.py in␣
↪_draw_list_compositing_images(renderer, parent, artists, suppress_composite)
    130     if not_composite or not has_images:
    131         for a in artists:
--> 132             a.draw(renderer)
    133     else:
    134         # Composite any adjacent images together

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\mpl_toolkits\mplot3d\axes3d.py in draw(self,␣
↪renderer)
    499                 # Then axes
    500                 for axis in self._get_axis_list():
--> 501                     axis.draw(renderer)
    502
    503             # Then rest

~\anaconda3\lib\site-packages\matplotlib\artist.py in draw_wrapper(artist,␣
↪renderer, *args, **kwargs)
     49                 renderer.start_filter()
     50
---> 51             return draw(artist, renderer, *args, **kwargs)
     52         finally:
     53             if artist.get_agg_filter() is not None:

~\anaconda3\lib\site-packages\mpl_toolkits\mplot3d\axis3d.py in draw(self,␣
↪renderer)
    219         renderer.open_group('axis3d', gid=self.get_gid())
    220
--> 221         ticks = self._update_ticks()
    222
    223         info = self._axinfo

~\anaconda3\lib\site-packages\matplotlib\axis.py in _update_ticks(self)
   1027         """
   1028         major_locs = self.get_majorticklocs()
-> 1029         major_labels = self.major.formatter.format_ticks(major_locs)
   1030         major_ticks = self.get_major_ticks(len(major_locs))
   1031         self.major.formatter.set_locs(major_locs)
```

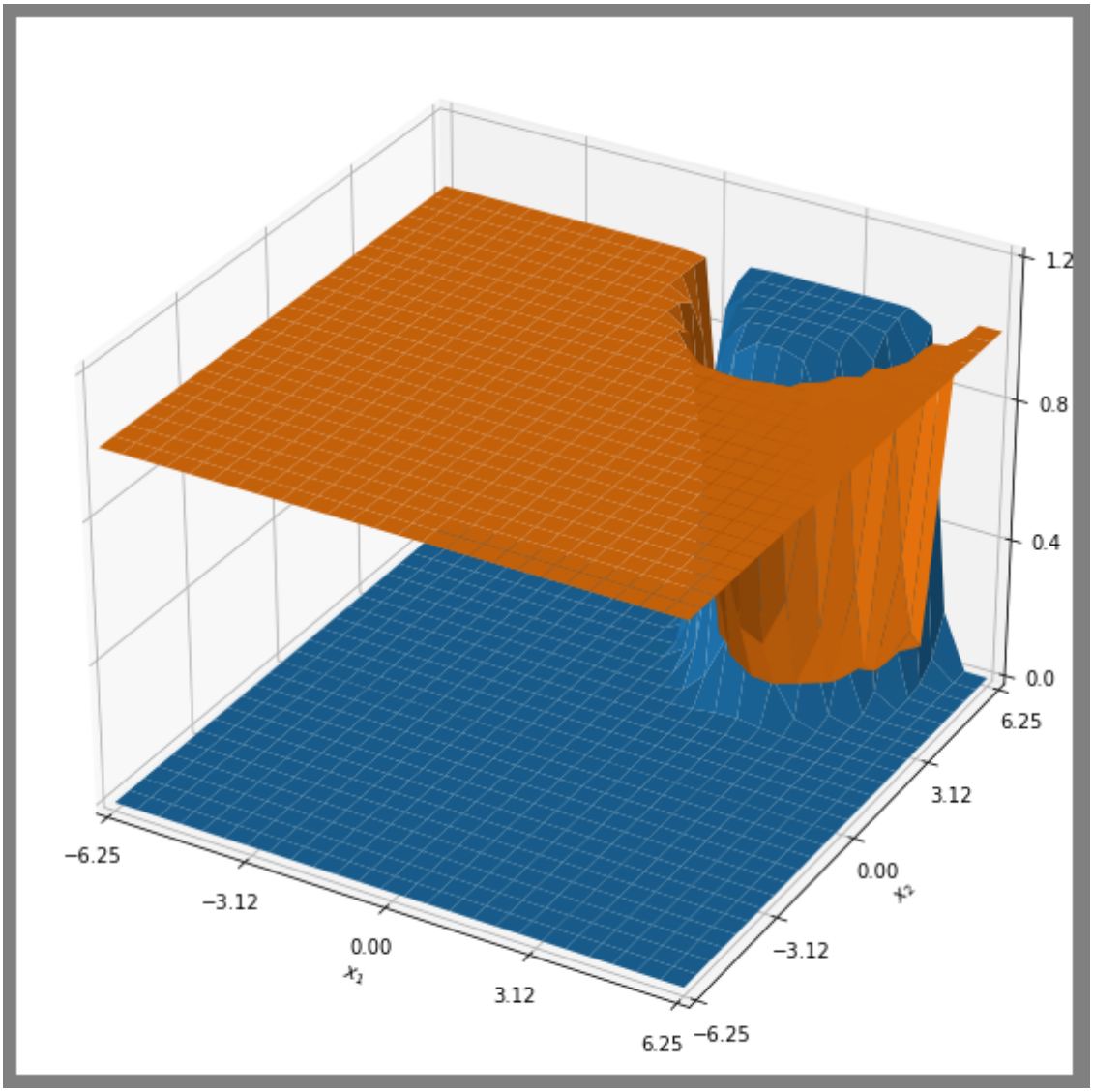
```
~\anaconda3\lib\site-packages\matplotlib\ticker.py in format_ticks(self, values
    260     def format_ticks(self, values):
    261         """Return the tick labels for all the ticks at once."""
--> 262         self.set_locs(values)
    263         return [self(value, i) for i, value in enumerate(values)]
    264

~\anaconda3\lib\site-packages\matplotlib\ticker.py in set_locs(self, locs)
    804                 self._compute_offset()
    805             self._set_order_of_magnitude()
--> 806             self._set_format()
    807
    808     def _compute_offset(self):

~\anaconda3\lib\site-packages\matplotlib\ticker.py in _set_format(self)
    900             # We needed the end points only for the loc_range␣
 ↪calculation.
    901             locs = locs[:-2]
--> 902         loc_range_oom = int(math.floor(math.log10(loc_range)))
    903         # first estimate:
    904         sigfigs = max(0, 3 - loc_range_oom)

ValueError: cannot convert float NaN to integer
```

```
<Figure size 720x720 with 1 Axes>
```

The KNN method expands V_n until R contains K_n samples, if we have less than K_n samples, then V_n tries to expand forever. In this case, K_n is not working for $K_n = 5$ because we only have 4 samples for one class and 3 for the other.

1.8 Section k)

[9]:
```
x1, x2, my, Sgm, posterior, df = labsol3()
classplot(posterior, x1, x2, 0, gsv={'gsv': 1, 'figstr': 'pdf'})
```



1.8.1 Student information

Álvaro Esteban Muñoz
a.estebanmunoz@stud.uis.no

Matthew Bwye Lera
264491@uis.no

Gero Kolhof
264546@uis.no