University of Stavanger

Stavanger, November 19, 2021

Laboratory exercise 2

ELE520 Machine learning

A PDF version of the report of the student solution to the exercise including figures and answers to questions shall be submitted on CANVAS.[1]

Problem 1

This problem is supposed to provide a visualisation of the usage of Bayes decision theory for a minimum error classifier.

We want to design a pattern recognition system that classifies 2-dimensional feature vectors $\{\mathbf{x}\}$ to class ω_1, or class ω_2.

It is known that the distribution between the two classes are $1/2$ and $1/2$ respectively for class ω_1 and class ω_2. Furthermore the feature vectors of the two classes are normally distributed around $\boldsymbol{\mu}_1 = (3\ 6)^t$ with covariance matrix

$$\boldsymbol{\Sigma}_1 = \begin{pmatrix} 1/2 & 0 \\ 0 & 2 \end{pmatrix} \tag{1}$$

for class ω_1, og around $\boldsymbol{\mu}_2 = (3\ -2)^t$ with covariance matrix

$$\boldsymbol{\Sigma}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \tag{2}$$

for class ω_2.

[1] If it is not possible to export the Jupyter notebook to PDF, the ipynb and py files can be submitted.

Trygve Eftestøl, Professor
Faculty of Science and Technology
Department of Electrical and Computer Engineering
Kjølv Egelands hus
University of Stavanger
N-4036 Stavanger.
Telephone: +47 51 83 10 00.
Telefax: +47 51 83 17 50.
E-mail: trygve.eftestol@uis.no
www.ux.his.no/~trygve-e

It is recommended to make a script, e.g. *labsol2.ipynb*, for the commands needed for solving the problems. As you work out this problem, you will realise that you will be reusing code. You might find the code listing of pseudo code for a proposed solution shown in listing 1 useful. You will also find some comments to the pseudo code at the end of this document. It is recommended to study this before starting working with the problems.

If you find working with iterations (for loops) and conditions (if) difficult, it is recommended to at least use the vector and list representation for the variables. This will make it easier to handle the iterations and conditions later on.[2]

a) Use *norm2D* to generate the class-conditional PDF for class ω_1 from problem 2 and plot it with red surface color (`col='r';`).

b) Use *norm2D* to generate the class-conditional PDF for class ω_2 and plot it with blue surface color, so that the two PDFs are shown in the same figure.

c) Repeat the two previous sub problems, but now for $P(\omega_i)p(\boldsymbol{x}|\omega_i), i = 1, 2$.

d) Identify the decision boundary and decision areas.

e) Change the a priori probabilities to 0.1 and 0.9 for class 1 og 2 respectively, and repeat the previous sub problem. What effect does thios have on the decision boundary?

   (If you do not find the change spectacular, you might first change the mean vectors of the two classes to $\boldsymbol{\mu}_1 = (3\ 2)^t$ and $\boldsymbol{\mu}_2 = (3\ 1)^t$. respectively)

f) Show the á posteriori probabilities, $P(\omega_i|\boldsymbol{x}), i = 1, 2$, for the two classes. Compare the decision thresholds and decision regions with the ones you found in the previous subtasks.

Comments to the pseudo code The main idea is to use python vectors and lists for the class specific functions which you can iterate over the classes. For example, if you let $P(\omega_i)$ and $P(\omega_2)$ be stored in the vector `Pw`, the values of $P(\omega_1)$ and $P(\omega_2)$ will be assigned to `Pw[0]` and `Pw[1]` respectively.

What happens inside the iterations (and conditions) are denoted by the level of indentation as it would be in the proper python code.

[2] If you handle this now, the remaining laboratory exercises will come at much less effort than if you do not!

As for the class specific density functions $p(\boldsymbol{x}|\omega_i)$ and parameters $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$, $i = 1, \ldots, M$ these should be represented as vectors and matrices. So python lists will be a practical choice for representing these.

Note that the number of classes is referred to as `M` in the pseudo code, but the program will not know it until line number 17, where it should determine its value from the `my` or `Sgm`.

Note that the choice of discriminant function is controlled by the variable `discr`. This will let you solve the problems where you are going to plot the various discriminant functions efficiently.

You might also note that some variables are hard coded in the function. You might consider setting these variables in your notebook and pass them in as variables instead.

Note that you can also make a function for plotting.

```
import getopt
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d
from pdffuns import *


def labsol2(discr='pxw'):

    # Initialise values:
    # - axes, x1 and x2
    # - 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
    # - determine number of classes, M
    # - initialise pxw as empty list
    # - initialise total density function, px as zero
    # - iterate over classes, i = 0,...,M-1
    #  - determine pxw[i] by using norm2D
    #  - update px

    # Determine discriminant functions, g[i], i = 0,...,M-1
    # - initialise g as empty list
    # - iterate over classes, i = 0,...,M-1
    #  - on condition of discr determine selected
   discriminant function
    #   - Scaled pdfs
    #   - Posterior probability
    #   - pdfs (not really discriminant functions)
    # - plot discriminant functions
```

Code Listing 1: labsol2.py