PROGRAMMING ASSIGNMENT 2 (HOMEWORK 3)
MA5755
DUE FEB 28

In this exercise we implement and test the Gaussian discriminant analysis, as described in Sec 9.2 in Murphy's book. The project can be broken into three tasks

(1) Generate the data points for all classes $c \in \{0, \dots C-1\}$, and compute the approximations $\hat{\mu}_c$ and $\hat{\Sigma}_c$ that are based on the data points. Also, generate a second set of points, with known tlabels, hat will be used to test the classification function.
(2) Evaluate the multivariate Gaussian distribution on the test points of all classes to obtain the probabilities $p(X = t|Y = c)$, where t runs through all test points and c runs through all classes.
(3) Compute the probabilities $p(Y = c|X = t)$ on the test points using Bayes' formula and for every test point find the value of c with the maximal probability.

In your implementation, you should write a separate subroutine for each task. These routines should be written such that they can be used for any dimension D and any number of classes C.

You should first experiment with artificially generated data. To that end, use the numpy routine `numpy.random.multivariate_normal` that generates points randomly by the multivariate Gauss distribution. In that case, the subroutine for task 1 has an interface as follows

```
#
# Parameters
#   mean  center of the Gaussian
#   cov   covariance matrix
#   nx    number data points
#   nt    number test points
# Returns
#   x   set of random vectors of length nx (data points)
#   t   set of random vectors of length nt (test points)
#   mu  mean based on x
#   Sgm covariance matrix based on x
#
def getArtificialData(mean, cov, nx, nt):
```

Note that this routine calls the `random.multivariate_normal` twice, i.e., for the data and test set. The vector `mean` and the matrix `cov` must be in $\mathbb{R}^D$ and $\mathbb{R}^{D\times D}$. The returned values `mu` and `Sgm` are computed by (9.19) and (9.20) in Murphy's book. They should be similar to `mean` and `cov`, but are different, in general. This routine must be called for each class c. To get the test points of each class into one long vector use the `numpy.concatenate` function.

The interface for the second task should look like this

```
#
# Parameters
#   t   points: t[n,d] is the d-th component of the n-th point
#   mu  mean
#   Sgm covariance matrix
# Returns
#   p  p[n] is the probability density function at the n-th point
def evaluateMultiVarGauss(t, mu, Sgm):
```

This routine must be called for each class c with the same merged test points `t` .

The interface for the third task should look like this

```
#
# Parameters
#    t   coordinates of the test points: t[n,d]
#    pXY conditional probabilities: pXY[n,c]
#    yEx exact labels of the test points: y[n]
def calculateTestSet(t, pXY, yEx: int):
```

Here `pXY` is a matrix whose columns are the output of `evaluateMultiVarGauss`. This routine should also print the probabilities and the actual and computed labels for each test point. It should also count the number of mislabelled points.

To test your routines generate three classes with the following parameters

$$\mathbf{mean} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \mathbf{cov} = \begin{bmatrix} 2 & 1 \\ 1 & 50 \end{bmatrix} \quad \mathbf{nx} = 40 \;\; \mathbf{nt} = 10$$

$$\mathbf{mean} = \begin{bmatrix} 7 \\ 5 \end{bmatrix} \quad \mathbf{cov} = \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} \quad \mathbf{nx} = 80 \;\; \mathbf{nt} = 20$$

$$\mathbf{mean} = \begin{bmatrix} -5 \\ 5 \end{bmatrix} \; \mathbf{cov} = \begin{bmatrix} 5 & 2 \\ 2 & 3 \end{bmatrix} \quad \mathbf{nx} = 20 \;\; \mathbf{nt} = 5$$

(1) Run the code as described above. This will generate quadratic decision boundaries. This should give a very small number of misclassified test points.

(2) Modify the code to test tied covariances (i.e., linear decision boundaries). You can compute the tied covariance matrix by the formula

$$\hat{\Sigma} = \frac{1}{N} \sum_c N_c \hat{\Sigma}_c$$

then you can re-use all previously written routines. Run the code with the same data as before and compare the number of misclassified points with the result you got in (1).

(3) Test the methodology on the iris data set, that was mentioned in the lecture. You can import it with the following commands

```
from sklearn.datasets import load_iris
iris = load_iris()
```

the petal/sepal dimensions are in the array `iris.data` and the corresponding labels are in the array `iris.target`. You will notice that the first 50

entries have label 0, followed by 50 entries with label 1 and 50 entries with label 2. Use the first 40 entries in each class as data points and the next 10 entries as test points. You have to write a new routine `getIrisData()` with similar functionality as `getArtificialData()`. If you have done the implementation of `evaluateMultiVarGauss()` and `calculateTestSet()` right, then you will be able to run these routines with the iris data set as well. Again, count the number of misclassified items in the test set.