GENERAL REMARKS The programming problems require you to fill the gaps in the provided MATLAB scripts. Use the problem description below and the comments in the code to solve the problems. The missing parts are typically indicated by '...' (not to be confused with line breaks!), where each '...' might be replaced by multiple lines of code. Following the provided code is recommended but only leads you to one of many possible solutions. If some part of the code seems unclear or counterintuitive to you, feel free to depart from it. Be aware, however, that the suggested variable names and structures are typically used later in the script, e.g. for plotting, and occur again in other scripts and problems. In any case, your code should produce the same output as the Musterlösung. **Note**: To be able to run the code, you must have the Statistics, Signal Processing, Image Processing, and Wavelet toolboxes installed, and you must have the folder `utility` and all subfolders as well as the location of the ECG/EEG datasets added to your MATLAB path.

Problem 1

In this problem, we will generate bivariate data from two normal distributions and classify the data using logistic regression. `runTest.m` executes the main part of the program and calls `modelFitVal.m` to fit and cross-validate the model. We will visualize the underlying distributions and the resulting decision boundary.

Part 1: Sampling and Analyzing

`runTest.m` begins by specifying the parameters for generating the data: sample size, means, and covariance matrix.

```
nSamples = 500;
mu1 = [1,3];
mu2 = [3,0];
sigma = [2.0 1.5
         1.5 2.0];
```

First, we generate the data using these parameters and an appropriate function, concatenate the two sample populations, and create a vector of labels (ones for one population, zeros for the other).

```
X1 = ...
X2 = ...
X = [X1; X2];
L = ...
```

`X` will be a 2*`nSamples`×2 matrix and `L` will be of length 2*`nSamples`.

Next, we want to see what the two underlying distributions look like along the two dimensions x_1 and x_2 (not to be confused with the two sample populations `X1` and `X2`). Create a grid of 100×100 equidistant points within the bounds of the sample populations, using `xVector`, and compute the two PDFs for each point.

```
xVector = linspace(min(X(:)), max(X(:)), 100);
...
pX1 = ...
pX2 = ...
pDist = reshape(pX1 + pX2, 100, 100);
```

You might want to use the `meshgrid` function and reshape the resulting matrices to vectors. If you have trouble with the suggested one-liners, use for-loops instead. The result `pDist` should be a 100×100 matrix containing the sum of both PDFs.

Now we train a generalized linear model on the sample data. Look into the MATLAB documentation for `glmfit` to find the right arguments for the distribution and link function.

```
coeff = glmfit(...);
```

The result `coeff` will be a vector of 3 coefficients.

To display the decision boundary of the fitted model, we solve the iso-error problem, i.e. we compute the line on which $P(C_1|x) = P(C_2|x) = 0.5$. This means solving

$$0.5 = \sigma(w^T x + w_0)$$

with σ being the logistic function; $w = (w_1, w_2)$; w_0, w_1 and w_2 given by `coeff`; and $x = (x_1, x_2)$. We have to rewrite the equation for x_2 and set x_1 to `xVector`.

```
x2db = ...
```

Now we validate the fitted model (find the appropriate function!) on the same 100×100 data points as before to get the posterior densities predicted by the model. (We only compute $P(C_1|x)$, since $P(C_2|x) = 1 - P(C_1|x)$.)

```
posteriorC1 = ...
posteriorC1 = reshape(posteriorC1, 100, 100);
```

Finally, we want to again train and 10-fold cross-validate the model, using our own function.

```
kCross = 10;
pCorrect = modelFitVal(X, L, kCross);
```

The remainder of `runTest.m` is about plotting and doesn't need to be edited.

Part 2: Classification

The function `modelFitVal.m` should take a data matrix (samples×dimensions), a correpsonding label vector, and the number of partitions, then iteratively divide the data and labels into training sets and test sets, fit and validate the GLM, and return the mean percentage of correctly predicted labels.

First, we create a random permutation of the sample indices, which we will use when composing the partitions.

```
nSamples = size(X, 1);
randIdx = ...
```

This way we avoid order effects on model performance but still create disjoint partitions. With our random data, order doesn't matter, but will in the later problems.

In the subsequent loop we divide the data and labels into test and training sets. From `randIdx` we select $\frac{\texttt{nSamples}}{\texttt{k}}$ indices for testing and the remaining $\frac{(\texttt{k}-1)\texttt{nSamples}}{\texttt{k}}$ indices for training.

```
pCorrect = 0;
for iPart=1:k
    idxTest = ...
    idxTrain = ...
    xTest = X(...);
    xTrain = X(...);
    lTest = L(...);
    lTrain = L(...);
```

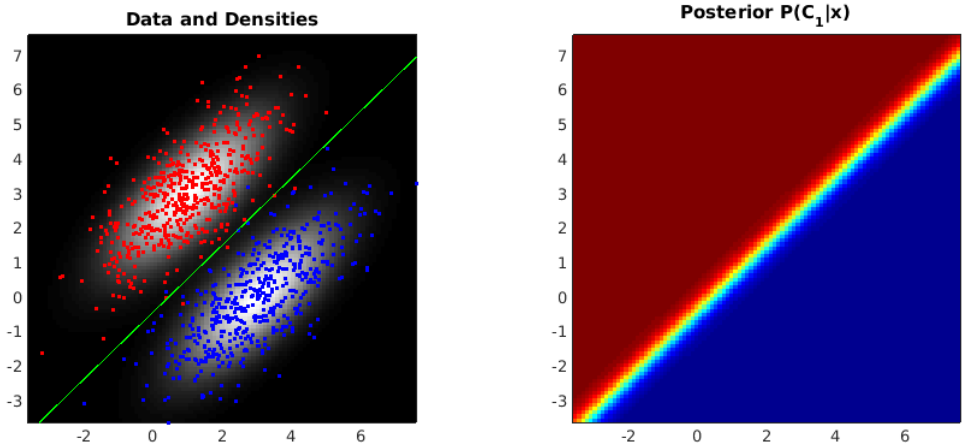
Note that `nSamples` need not be a multiple of `k`. Make sure that the size of test sets (same for training sets) does not differ by more than 1 across partitions. Make also sure that all samples are used for testing exactly once.

Now we fit the logit model as before and compare the rounded validation output, i.e. the predicted labels for the test data, to the actual labels. The percentage of correct predictions is then accumulated over iterations.

```
    coeff = glmfit(...);
    lPredicted = round(...);
    pCorrect = pCorrect + 1/k*mean(lPredicted==lTest);
end
```

Part 3: Plotting and Exploring

Your code should print the model's average classification performance and produce the two figures below, showing the sample data with their underlying distributions and the model's decision boundary, and the class posterior probabilities, respectively.



Note that the plots will vary a bit at between runs because of the randomness in sampling and cross-validation. Classification performance should be at or near 100 %. You may want to try different distribution parameters and sample sizes and see how performance and plots change. Try to understand the warning messages you might get.