1 Practical: Learning Vector Quantization

In Nestor you will find the file `data_lvq.mat`, which contains 100 two-dim. feature vectors. Here, we assume that the first 50 points belong to class 1, the other data points belong to class 2.

1.1 Implementation (3 points)

Implement the LVQ1 algorithm as introduced and discussed in class, using simple (squared) Euclidean distance. Consider
(a) LVQ1 with one prototype per class
(b) LVQ1 with two prototypes per class.

Your code should have the following structure:

- Read in the file containing the data, define the corresponding class labels and determine the following:
  N: the dimension of input vectors and *P*: the number of examples
- Set the following parameters

  K: the number of prototypes,

  η: the learning rate (constant step size),

  t_{max}: maximum number of epochs (sweeps through the data set)
- Initialize each prototype by random selection of a data point from the corresponding class
- Repeat for epochs $t = 1$ to $t = t_{max}$:
  - In each *epoch* present all of the data set in randomized order (every example is presented exactly once, the order of examples is different in every epoch). In matlab this could be done conveniently by permuting the examples with the `randperm(P)` command. Take care that also the training labels are permuted in the very same way!
  - perform an epoch of training using all of the P examples. At every individual step present a single example to the system, evaluate the distances from all prototypes and update the *winning* prototype according to the LVQ1 prescription.

After each epoch, determine the number E of misclassified training examples, i.e. the "training error". Plot $E/100$ (the error in %) as a function of the number of epochs (learning curve) and stop the training when E becomes approximately constant. It turns out that for this data set a small learning rate of 0.002 appears appropriate. With this learning rate you should get reasonable results after, say, 100 or 200 epochs.

1.2 Perform training and report results (7 points in total)

You should hand in a report comprising

- (1 point) a brief introduction in words
- (2 points) one example learning curve for (a) and (b) each (as figures with appropriate labelling and caption)
- (2 points) a plot which shows the data with LVQ1 labels (e.g. as black dots vs. white circles) in the two-dim. feature space for case (b) at the end of the training process; also mark the prototype positions
- (2 points) a brief discussion of your results

Remarks:

- If you apply special ”tricks” in your code explain them in the text and provide the corresponding lines of code as well
- Of course you should code LVQ1 yourself, do not use functions from the matlab neural network toolbox or code retrieved from some repository

Bonus (suggestions)

1 point max. in total:

- initialize prototypes in (or very close to) the class-conditional mean vectors and compare the learning curves with those of the random initialization
- consider systems with three or four prototypes per class
- display the ’learning trajectory’ of prototypes in the two-dim. space
- implement and run ”GLVQ” which, for every single example, updates the closest correct and the closest wrong prototype
- run your LVQ implementation on the 3-class Iris data set, which is available at `https://archive.ics.uci.edu/ml/datasets/iris`