# Assignment: Learning Vector Quantization (LVQ1)

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, while the other data points belong to class 2. Implement the LVQ1 algorithm as introduced and discussed in class, using simple Euclidean distance. (Obviously you could start from your VQ code for the previous assignment.)
Consider
(a) LVQ1 with one prototype per class
(b) LVQ1 with two prototypes per class
Initially, place the prototypes in randomly selected data points (from the corresponding correct class).

The structure of the code remains essentially the same, but of course you have to take into account the labels of data and prototypes. An important note: You should shuffle the order of data randomly before each epoch, as in the VQ exercise. Of course you have to make sure that the labels remain consistent, i.e. they are shuffled in the exact same way.
Use all of the data in the LVQ1 training process repeatedly. 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 it 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.
**You should hand in (at least):**
- a brief introduction
- one example learning curve for (a) and (b) each
- a plot which shows the 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
- a brief discussion of your results
- your Matlab code, if you apply special "tricks" explain them in the text and provide the corresponding lines of code as well.

**Remark:**
- of course you should code LVQ1 yourself, do not use functions from the neural network toolbox etc. or code retrieved from some other repository

--------------------

**Suggestions** for possible extensions:
- 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 'trajectory' of prototypes in the two-dim. space
- implement "LVQ2" which, for every single example, updates the *closest correct* and the *closest wrong* prototype by an LVQ1-like update step