Machine Learning (lecture) Fall 2014: Exercise sheet 2

Exercise. *General idea*: continue working with your solution to exercise 1, that is, re-use the feature that you invented for classifying "zero" vs. "one" digit images. Then use cross-validation to find the best model capacity for the bin-counting classifier.

Details. Again use the data from exercise 1, that is, 200 images each of "zeros" and "ones". Split this into a set of 100 images each, and use only the first 100 images for training your model. Pretend that you simply don't have access to the remaining 100 images of each class. Call the data constituted by the first 100 images D (that is, D contains 100 "zero" and 100 "one" pics). Use 10-fold cross-validation for optimizing model capacity. For a bin-counting based classification model (based on the feature that you invented in exercise 1) the model capacity increases with the number of bins. Compute a sweep through increasing model capacities (i.e., through using increasing numbers of bins), and for every capacity compute the average training error and the average validation error (average taken over the 10 folds of cross-validation). Plot these two errors against the number of bins, obtaining a figure that should look similar to Figure 2.9 from the lecture notes. Select the capacity (= nr of bins) where the validation error is minimal. Then train a model with this capacity on all of D. Finally, pretend that now suddenly you are given the other 100 pics from each class (the ones that you pretended not to know about so far). Compute the test error of your model on these.

Deliverables:

1. A Matlab or Python function `optimize_capacity` which takes as input the data D and returns the optimal model capacity (= optimal nr of bins) in the form of an integer k. As a side effect, calling this function generates a plot similar to Figure 2.9.
2. Another function `compute_model` which takes as input the data D and an integer k indicating the number of bins, and returns a binary vector V of length k and a real-valued vector L of length k–1. In its i-th position, vector V has a value of 1 iff a feature value falling into the i-th bin leads to a classification as "zero" (class 1; class 2 is the "ones" class). The vector L gives the limits of the bins (the first bin extending to minus infinity, the last to plus infinity, hence k–1 limits give k bins).
3. Another function `classify` which takes as input a binary vector V of some length k, a bin-limit vector L of length k–1, and a feature value F. The function returns an integer output 1 or 2, depending on whether the bin in which F falls is assigned to class 1 or 2 by V.
4. A function `myfeature` which takes as input a 240-sized image vector and returns the value of "your" feature.

Notice that the following little script (using Matlab notation)

```
[V, L] = compute_model(D, optimize_capacity(D));
classDecision = classify(V, L, myfeature(testpic))
```

will optimize capacity, generate a model (= V and L) of optimized capacity, and use that model to classify a test picture. The final deliverable should be a complete script:

5. A script `complete_experiment` which can be conveniently called by Dima or myself and uses all your functions from deliverables 1.-4. to run through the entire model learning suite and classifies a test picture.

If you wish, you may experiment with slightly more complex models where the bins may have individually different widths. Makes the cross-validation search for an optimal-capacity model much harder but should eventually lead to an even better model (fewer errors on the test data).

Send your functions and the script in a zipfile to h.jaeger@jacobs-university.de and the course TA Dzmitri Bahdanau d.bahdanau@jacobs-university.de by Sunday, September 28, midnight. Please name your zipfile as follows:
`<your last name>HW2.`