Pattern Recognition Laboratory – Sessions #3 & #4
Recognition of handwritten digits with linear classifiers

Due date: 6.05.2024 (LAB/101)

In this assignment, your task will be to recognize handwritten digits. We'll use one of the most widely used data sets in the pattern recognition: MNIST handwritten digits (http://yann.lecun.com/exdb/mnist). Training and test sets can be downloaded at the address given above. In addition they are available on the Galera server (http://galera.ii.pw.edu.pl/~rkz/epart/mnist.zip).

You should know that the images are normalized after scanning as follows:

1. The rectangle containing the black-and- white image of a character scanned at a resolution of 300 dpi is scaled proportionally to fit into square 20 by 20 pixels. During scaling an image is converted to grayscale.
2. The center of gravity of the scaled character is determined, and character is placed in a 28x28 pixel image so that the center of gravity lie in the middle of the bigger image.

Your task is to produce a classifier that uses **linear classifiers** distinguishing individual digits. In addition to the quality of classification on the test set you should produce **confusion matrix**. Additional task is to propose a different (let's hope - better) method for determining the classifier decision than simply voting of elementary classifiers.

The point of reference is the classic voting (45 linear ovo, i.e. *one versus one*, classifiers) classifying pairs of digits if it collected maximum possible number of votes = 9. In other cases, voting classifier makes reject decision. The tests used the first 40 principal components. Classification results are summarized in the table below (although an interesting insight into the classification may give confusion matrix analysis).

	MNIST Training Set			MNIST Testing Set		
	OK.	Error	Rejection	OK.	Error	Rejection
Classification coefficients	91.00%	6.29%	2.71%	91.03%	6.24%	2.73%

The task can be divided into a few parts:

1. Preparation of the function to compute separation plane parameters given a training set containing just two classes - `perceptron`. The easiest way to accomplish it is to use two-dimensional data sets , which can be visualized together with the separating plane.
2. Checking the algorithm for multidimensional digits data. You should store individual *one vs. one* classifiers quality to put them in your report. Because you'll have to check expansion of the features reduce dimensionality of data with PCA (40-60 primary components).
3. *Canonical* solution is 45 voting classifiers - one for each pair of digits - and making the final decision with unanimity voting (only digits with 9 votes are classified; if the number of votes is smaller classifier produces reject decision). You should report individual classifier error rates as well as quality of the ensemble. Quite interesting insight into ensemble operation can give you a confusion matrix.

4. Preparing one-versus-rest ensemble of classifiers. Here you'll have to think about organizing the training (a function similar to `trainOVOensemble`). Even before, you should decide about representation of classifiers in the matrix; with code reuse in mind it would be wise to put in the negative class label in the second column. Such a value that will allow you – with minimal changes in code – to use `voting` and `unamvoting` functions. Alternative is of course to write separate functions for OVR ensemble.
5. Check the quality of the OVR ensemble using full training and testing sets. Include individual classifiers error coefficients together with the final digit classification quality.
6. Now expand features with `expandFeatures` function. This function adds new following features: $x_i x_j \; for \; i \leq j$. It's important to know the number of new features: if F is the number of original features $F + \frac{F(F-1)}{2}$ new features are created. For 40 primary components this gives 40 + 780 new features (860 in total).
7. Repeat points 3-6 with this expanded data set.
8. What are the differences in individual classifiers' error coefficients (compare the results for original and expanded features). What's the difference in final digit classification quality?
9. Extra task for the last two points. Can you think of a solution which is better than any of the standard approaches you implemented in previous points? Implement it and compare the result with the best (up to this point) digit classifier.
10. Compare confusion matrices for both solutions (these two confusion matrices should be included in the report). Please, limit the number of confusion matrices in the report to just two.

Your report should include:

1. Error rates of individual basic classifiers in canonical OVO solution (for original & extended data set). Digit classification results for both cases.
2. Error rates of individual basic classifiers in OVR solution (for original & extended data set). Digit classification results for both cases.
3. Error rates of individual basic classifiers in OVR solution trained on balanced positive/negative class training set (for original & extended data set). Digit classification results for both cases.
4. The method you developed to improve the classification quality. Comparison of the digit classification results and confusion matrices for the best standard and your non-standard solution.
5. You should upload your report in pdf format together with the code but **you should not send the data sets (I hope that size limit set in LeOn prevent this)**.

Those interested can check what classification results your classifiers achieve on a data set prepared locally (i.e. in Warsaw). The data set in MNIST format is available at: http://galeranew.ii.pw.edu.pl/~rkz/rob/pldigits.zip. I explained the effects of using the training set from a different population than "production" data, but it is worth seeing this with your own eyes.

List of files in the starter pack:

`compErrors.m`	- computes classification coefficients from the confusion matrix
`confMx.m`	- creates confusion matrix using ground truth data and classification results
`expandFeatures.m`	- computes new set of features multiplying original feature values
`mainscript.m`	- main experiments notebook
`pcaTransform.m`	- applies linear transformation to feature values
`perceptron.m`	- computes separating hyperplane with perceptron algorithm
`prepTransform.m`	- computes PCA transformation for selected number of primary components
`readmnist.m`	- reads images and labels in MNIST format
`readSets.m`	- reads training and testing sets from MNIST database (uses `readmnist`)
`trainOVOensemble.m`	- training set of classifiers for one vs. one classifier ensemble
`trainOVRensemble.m`	- training set of classifiers for one vs. one classifier ensemble
`trainSelect.m`	- executes several times perceptron and selects the best solution (with smallest number of errors)
`unamvoting.m`	- unanimity voting classifier
`voting.m`	- fills ensemble classifiers vote table (matrix)