LAB 2: Spectral filters & multi-class classification

- This laboratory is about binary and multi-class classification and model selection on synthetic as well as real data, focusing on the role and on the properties of spectral filters.
- The aim of the lab is to play with the libraries and get a practical grasp of what we have discussed in class.
- Follow the instructions below.

Goal:
This lab is divided in three parts depending of their level of complexity (**Beginner**, **Intermediate**, **Advanced**). Your goal is to entirely complete at least one of the three parts.

Setup instructions:
Running OCTAVE Download the file `regml2020_lab2.zip` from the syllabus on the course website (`http://lcsl.mit.edu/courses/regml/regml2020/#syllabus`), extract it and add all the sub-folders to the OCTAVE path. This file includes all the code you need!

PART I: Beginner

Overture: Warm up

Open the files `loadDataset_octave.m` and `learn_octave.m` in Octave. Look at the code, and see that the first lines are dedicated to the set up of various options and parameters. The objective of this section is to familiarize you with the code.

- **I.A** In `loadDataset_octave.m`, look at the parameters and modify them to generate a simulated classification dataset of type *Spiral*. To do so, after changing the parameters and saving `loadDataset_octave.m`, execute it directly from Octave's interface, or type `loadDataset_octave;` in the shell.
  Tip: If too many figures are opened at the same time, you can close them all at once with the command `close all`.
- **I.B** Use `learn_octave.m` to find a classifier for this dataset. For this, take a *Truncated SVD* filter, associated to a *Gaussian* kernel. Using a Gaussian Kernel requires to select a parameter `sigma`, let the code choosing it automatically. Have a look at the parameter selection part and the various options of KCV. To select the regularization parameter you can either choose KCV or set a fixed value. To run the learning algorithm, save `learn_octave.m`, and execute it directly from Octave's interface, or type `learn_octave;` in the shell. Observe then the plot of the KCV error and the balance between training and test errors. Also have a look at data set, where a separation function has now appeared.

> **NOTE:**
> Remember that in the MATLAB code the regularization parameter for the *Tikhonov* (`rls.m`), *T-SVD* (`tsvd.m`) and *cut-off* (`cutoff.m`) is called `t` instead of λ. For the *NU-Method* (`nu.m`) and the *Landweber-Method* (`land.m`), the regularization parameter `t` is the number of iterations, that is, roughly speaking, `t` $\sim \frac{1}{\lambda}$.

Allegro con brio: Analysis

Carry on the following experiments either using `loadDataset_octave.m` and `learn_octave.m`, when it is possible, or writing appropriate scripts.

- **I.C** Back on the shell, check the content of directory `./spectral_reg _toolbox`. There you will find, among others, the code for command `learn` (used for training), `patt_rec` (used for testing), `kcv` (used for model selection on the training set). For more information about the parameters and the usage of those scripts, type:

```
help learn
help patt_rec
help kcv
```

Finally, you may want to have a look at the content of directory `./dataset_scripts` and in particular to file `create_dataset.m`, that allows you to generate synthetic data of different kinds.

- **I.D** Generate noisy data of *Spiral* type. Consider three algorithms, namely *RLS*, *Truncated SVD* and *NU-Method*. Observe how the training and test errors change as:
  - We change (increase or decrease) the regularization parameter `t`.
  - The training set size grows (try various choices of `n` as long as your computer supports you!).
  - The amount of errors in the labels in the generated data grows.

  Run training and testing for various choices of the suggested parameters.

- **I.E** Leaving all the other parameters fixed, choose an appropriate range for the regularization parameter, `tval=[t_min:t_step:t_max]`, and plot the training and the test errors for each `t`. For doing this, you might use the functions that you saw in I.C:

```
[Xtr, Ytr] = create_dataset(..);
[alpha, err] = learn(.., .., .., t, Xtr, Ytr, 'class');
training_error = cell2mat(err);

[Xtest, Ytest] = create_dataset(..);
[~,test_error] = patt_rec(..,..,alpha, Xtr, Xtest, Ytest, 'class');
```

  Use the KCV option to select by cross-validation the optimal regularization parameter, and see how it relates to your previous plot.

- **I.F** Leaving all the other parameters fixed, choose an appropriate range for the number of points in the training set, `nval=[n_min: n_step:n_max]`, and plot the training and test errors. What do you observe as `n` $\to \infty$? How do the different regularization parameters affect the learning process? Which are the main differences in terms of regularization between the methods?

PART II: Intermediate

Crescendo: Advanced Analysis

Carry on the following experiments either using `loadDataset_octave.m` and `learn_octave.m`, or writing appropriate scripts. In this part you have to focus more on the effects of regularization and on the correct choice of `sigma`.

- **II.A** Consider *gaussian-RLS* and perform parameter tuning in this case. This time, together with the regularization parameter `t`, you'll have to choose an appropriate `sigma`, the kernel parameter.
  - Try some (`sigma`, `t`) pairs and compare the obtained training error and test error.
  - Fix `t` and observe the effect of changing `sigma`.
  - Fix `sigma` and observe the effect of changing `t`.
  - Do you notice (and if so, when) any overfitting/oversmoothing effects?

  If you have time, you can try to plot the test error for a range of couples (`sigma`, `t`).
- **II.B** Compare *RLS* with *NU-Method* on a kernel of your choice.
  - Tune the parameters by using KCV.
  - Compare the time needed to obtain a solution.
  - Compare the training and test errors.

PART III: Advanced

Finale: The Challenge

The challenge consists in a learning task using a real dataset, namely *USPS*. This dataset contains a number of handwritten digits images. The problem is to train *the best classifiers* that are able to discriminate between digits `3`, `8` and `0`.

Have a look at the script `demo_lab2.m`. This script contains a code snippet to perform a multi-class classification task using the previously presented scripts (see I.C).

You should understand what the scripts are supposed to do, and train the classifiers in order to perform One vs. All classification for all the combinations of the digits `3`, `8` and `0`.

Once the classifiers have been trained, the model must be exported in a matrix file by means of the `save_challenge_lab2.m` script (to see how to use it please try the command `help save_challenge_lab2`.

Submission: You should drop your results in a matrix file named `name-surname` to the link: `https://www.dropbox.com/request/DWcrqDPQEulg7Dh0vqlz` by the end of the challenge session.
The results will be presented during the next class. The score is based on the accuracy of the classifier on a *completely independently sampled* test set.

Deadline: 6:00 PM.