Regularization Methods for Machine Learning

LAB 1: Binary classification and model selection

- This lab addresses binary classification and model selection on synthetic data.
- The aim of the lab is to play with the libraries and to 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 complete entirely, at least, one of the three parts.

Setup instructions:
Running OCTAVE

Download the file `regml2018_lab1.zip` from the syllabus on the course website (`http://lcsl.mit.edu/courses/regml/regml2018/#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 *Linear*. 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. See how the number of wrong labels affects the data set. You can also look at the other types of datasets, which are nonlinear.
  Hint: If too many figures are opened at the same time, you can close them all at once with the command `close all`.
- **I.B** From now, consider a *Linear* dataset, with 100 (resp. 1000) training (resp. test) samples, and some errors in the labels. Use `learn_octave.m` to find a classifier for this dataset. For this, take a *regularized least squares* filter, associated to a *linear* kernel, and use cross-validation to select the regularization parameter. 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:
In the code we use a different notation from the one you have seen in the classes. In the *Regularized Least Squares* method (`rls.m`), the regularization parameter is `t` instead of λ.

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 in 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 *Linear* type. Considering *linear-RLS*, 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 need the function `learn` that you saw in I.C:

  ```
  [alpha, err] = learn('lin', [], 'rls', t, X, Y, 'class');
  ```

  In this context, you can have access to the training and test errors corresponding to the parameter `t` with:

  ```
  training_error = cell2mat(err);
  Y_learnt = kernel('lin', [], Xt, X) * cell2mat(alpha);
  test_error = learn_error(Y_learnt, Yt, 'class');
  ```

  Use the KCV option to select by cross-validation the optimal regularization parameter, and see how it relates to your previous plot. If you could, would you use the test error to select the parameter?

- **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` $\to \infty$?

PART II: Intermediate

Crescendo: Advanced Analysis

- **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** Consider *polynomial-RLS* and perform parameter tuning as in II.A. How does the choice of the kernel affect the learning behavior of the algorithm? In particular, compare the performances of the polynomial and Gaussian kernels on the *spiral* and *moons* datasets with respect to the number of examples in the training set (e.g. `[10, 20, 50, 100, 1000]`) and the value of the parameter regularization (e.g. `[0.5, 0.1, 0.01, 0.001, 0.0001]`).

PART III: Advanced

Finale: The Challenge

The challenge consists in a learning task using a real dataset, namely *USPS (United States Postal Service)*: This dataset contains a number of handwritten digits images. The problem is to train *the best classifier* that is able to discriminate between the digits $\boxed{1}$ and $\boxed{7}$.

The data can be found in the `data` folder at the root of the git repository: load the files `one_train.mat` (this should be labeled as -1), and `seven_train.mat` (which should be labeled as $+1$) using the matlab `load` function.

Once the classifiers are trained, they must be exported by means of the script `save_challenge_lab1` (to see how to use it, look at its code). The file `demo_lab1.m` contains a code snippet to perform a simple binary classification task by means of the previously presented scripts.

Submission: You should drop your results in a matrix file named `name-surname` to the link: `https://www.dropbox.com/request/K4vX1jIOGiX8hks6mtJF` by the end of the challenge session.
The results will be presented during the next class. The score of your result

will be based on the accuracy of the classifier on a *completely independently sampled* test set.

Deadline: 6:00 PM.