Regularization Methods for Machine Learning

LAB 3: Sparsity-based learning

- This lab is about feature selection within the framework of sparsity based regularization, using elastic net regularization.
- 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 two parts depending of their level of complexity (**Beginner**, **Intermediate**). Your goal is to complete entirely, at least, one of the two parts.

Setup instructions:
Running OCTAVE Clone or download the course repository at `https://github.com/LCSL/RegML`. Add all the subfolders to the Octave path. The repo includes all the files you need!

Toy problem

We focus on a regression problem where the target function is linear. We will consider synthetic data generated (randomly sampled) according to a given probability distribution and affected by noise. You will have the possibility of controlling the sizes of training and test sets, data dimensionality and number of relevant features.

NOTE:
In the code we use a different notation from what you have seen in the classes. The minimized functional is:

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{2n}\|X\beta - Y\|^2 + \texttt{L2\_par}\frac{|||X|||^2}{2n}\|\beta\|_2^2 + \texttt{L1\_par}\|\beta\|_1,$$

where $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^n$.

In addition, in some scripts the sparsity parameter `L1_par` can be also referenced as `tau`, and the smoothing parameter `L2_par` as `smooth_par`.

PART I: Beginner

Overture: Warm up

Open the files `loadSparseDataset_octave.m` and `learnSparse_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 `loadSparseDataset_octave.m`, look at the parameters and generate a training set with the default parameters. Run then `learnSparse_octave.m`, with the default parameters, to start a training phase. The first figure displays the cross validation error together with the test error. The second figure displays the sparsity of the reconstructed signal depending on the chosen `L1_par`. Compare the sparsity of the reconstructed signal with the original one.
  Note: In this lab we perform regression and not classification. So the error is not a percentage error on the labels, but is measured with quantities of the form $(1/n)\|X\beta_{pred} - Y\|^2$.
- **I.B** Train your dataset by changing now the values for `L1_par` and `L2_par` in `learnSparse_octave.m`.
  - **I.B1** With `L2_par` $= 0$ and playing with `L1_par`, try to obtain a sparser or denser solution. How does the sparsity behave with respect to `L1_par`?
  - **I.B2** Repeat the experiment with a `L2_par` > 0. How do test error and number of selected features vary?
- **I.C** Repeat the experiments of I.B1, by considering this time a noisy dataset (you can take a noise parameter equal to 0.3). How does now behave the sparsity? What can you say about the training error?

Allegro con brio: Analysis

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

- **I.D** Back on the shell, have a look to the content of directory `./PROXIMAL_TOOLBOXES/L1L2_TOOLBOX`. There you will find, among others, the code for `l1l2_algorithm` (used for variable selection), `l1l2_kcv` (used for model selection with kcv or loo), `l1l2_pred` (for prediction on a test set).

  For more information about the parameters and the usage of those scripts, use the help.

Finally, you may want to have a look at file `l1l2_demo_simple.m` for a complete example of analysis. You might use it as a basis for writing your own script(s).

- **I.E** *(Prediction and selection)* Considering elastic net regularization, we want to observe, in a systematic way, how the training and test errors are sensitive to the parameters, as well as the number of relevant features of the solution (changing only one parameter at a time). To achieve this, you should try to pick some parameters, or run iteratively over a range of parameters, by exploiting the code contained in `l1l2_demo_simple.m`.
  - **I.E1** When we change the regularization parameter `L1_par` associated with the ℓ_1-norm.
  - **I.E2** When we change (increase or decrease) the regularization parameter `L2_par` associated with the ℓ_2-norm.
    Hint: Try the following fixed parameters: 20 points of dimension 100, with 15 relevant features, a noise level equal to 1 and `L1_par`=0.1.
  - **IE.3** The training set size grows (this is not the same than generating different training sets with an increasing size!).
    Hint: Try the same parameters as above, with `L2_par`=0.
  - **I.E4** The amount of noise on the generated data grows (the test set is generated with the same parameter of the training).
- **I.F** *(Large p and small n)* Perform experiments similar to those above changing p (dimensionality of the points), n (number of training points) and s (number of relevant variables). In particular, when $p \gg n$, look at how the result behave, depending whether $s < n$ holds or no. (try e.g $n = 80$ and $p = 300$). Try to identify different regimes.

PART II: Intermediate

Crescendo: Data standardization

From now you can only use your own scripts.

Import the classification dataset `part3-data.mat` given in the `data` folder at the root of the repo, which contains two matrices: `X` and `Y`. For this you can just double click on it in the file explorer, or otherwise write

```
temp=load('../../data/part3-data.mat'); X=temp.X; Y=temp.Y;
```

Here $X \in \mathbb{R}^{n \times p}$ is a matrix containing n points in $\mathbb{R}^p$, and $Y \in \{-1; +1\}^n$ is generated from $Y = X\beta$, where $\beta \in \mathbb{R}^p$ is a sparse vector, meaning that it has s nonzero components, with s being small.

- **II.A** You do not know how many relevant features generated this data. Try to get an estimation of this number s, by using `l1l2_learn`, according to your observations in part I.
- **II.B** An other way to try to estimate s is to measure the correlation between the columns of X and Y. Indeed, the zero coefficients in β will ignore the corresponding columns in X while generating Y. To do so, you might use
  `c=abs(corr(X,Y)); stairs(c);`
  If you do not see which columns of X are the more correlated with Y, you can use `[~,I]=sort(c);` to sort its values and identify the more relevant features. You should have now a precise idea of what is s. Can you also identify which are these s features?
- **II.C** Use again `l1l2_learn` and tune the sparsity parameter `L1_par` in view to select only s features (s being your estimation from the previous question). Look at which are the selected features in this solution. Do they correspond to the ones you identified in II.B? If no, can you figure out why?