LAB 1: Local Methods for Classification

- This lab is about local methods for binary classification on synthetic data.
- The goal of the lab is to get familiar with the k-Nearest Neighbors algorithm (kNN) algorithm and to get a practical grasp of what we have discussed in class.
- Follow the instructions below. Think hard before you call the instructors!

Setup instructions:
Running OCTAVE or MATLAB

Download the file `regml2017_lab1.zip` from the syllabus on the course website (`http://lcsl.mit.edu/courses/regml/regml2017/#syllabus`), extract it and add all the sub-folders to the OCTAVE/MATLAB path. This file includes all the code you need!

PART I: Warm up with data generation

Open the file `MixGauss.m` in Octave/Matlab.

- **I.A** The function `MixGauss(means, sigmas, n)` generates dataset `[X,Y]`, where `X` is composed of mixed classes, each class being distributed according to a Gaussian law, with given means and variance. The points in the dataset `X` are numerated from `1` to `n`, and `Y` represents the label of each point. Have a look at the code or, for a quick help, type `help MixGauss` in the shell.
  Hint: if the command `help MixGauss` fails, this means that you haven't set up correctly your current directory.
  Hint: in Octave, to quit the help mode, type `q`.

- **I.B** Type in the shell the following commands:

```
[X, Y] = MixGauss([[0;0],[1;1]],[0.5,0.25],80);
figure(1); title('dataset 1');
scatter(X(:,1),X(:,2),50,Y,'filled');
```

  You can type `help scatter` to see what the parameters mean.

- **I.C** Now, you are going to generate a more complex dataset, following the instructions below. This dataset will be referred hereafter as *training dataset*.

- Call `MixGauss` with the appropriate parameters

  ```
  [Xtr,Ytr]=MixGauss(....)
  ```

  to produce a training dataset with four classes: the classes must live in the 2D space, be centered on the corners of the unit square (0,0), (0,1), (1,1), (1,0), and all with variance 0.3. The number of points in the dataset is up to you.

- Use the function `scatter` to plot this dataset.

- Transform the data into a 2-class problem, where the data on *opposite corners* are labeled the same, and where these labels are -1 and $+1$. Do this by just modifying `Ytr`, and verify that you did it well by plotting the data.
  Hint: if you produced the data following the centers order provided above, you may do: `Ytr = 2*mod(Ytr,2)-1;`. Otherwise, you may also use operations of the form `Ytr(Ytr==4)=-1;`.

- **I.D** Following the same procedure as in section I.C, generate a new set of data `[Xte,Yte]`, hereafter called *test dataset*, following the same distribution, and labeled in the same way in $\{-1, +1\}$.

PART II: The kNN classifier

The k-Nearest Neighbors algorithm (kNN) assigns to a test point the most frequent label among its k closest neighbors.

- **II.A** Have a look at the code of the function `kNNClassify`, by looking either in `kNNClassify.m` or typing `help kNNClassify` in the shell.

- **II.B** Use `kNNClassify` on the previously generated 2-class data from section I.C and I.D. Pick yourself a "reasonable" `k`. Below we propose three ways of evaluating the quality of the prediction made by the kNN method. Try each of them, and see the influence of the parameter `k` in the quality of the prediction.

  - **II.B1** Evaluating the prediction (plotting the data). Plot the test data `Xte` twice, once with its true labels `Yte`, and once with the predicted labels `Ypred`. A possible way to do:

    ```
    figure;
    scatter(Xte(:,1),Xte(:,2),50,Yte,'filled');
    hold on;
    scatter(Xte(:,1),Xte(:,2),70,Ypred,'o');
    ```

- **II.B2** Evaluating the prediction (measure the percentage error). Compare the predicted labels with the true ones. A possible way to do:

  ```
  sum(Ypred~=Yte)./size(Yte,1)
  ```

- **II.B3** Evaluating the prediction (visualizing the separating function). To visualize the separating function (and thus get a more general view of what areas are associated with each class), use the routine `separatingFkNN`. You may use `help separatingFkNN` in the shell, or look directly at the code.

PART III: Parameter selection - What is a good value for k?

So far we selected manually the parameter `k`. Now we want an automatic policy to choose it.

- **III.A** Perform a hold out cross validation procedure on the available training data. You may want to use the function `holdoutCVkNN` available in the zip file. Plot the training and validation errors for the different values of `k`.
- **III.B** Add noise to the data by randomly flipping the labels on the training set, and call it `Ytr_noisy`. You can use the function `flipLabels` to do that. How does the validation error behave now with respect to `k`?
  Note: Keep track of the best `k`, and the corresponding validation error. You will need it in III.D.
- **III.C** What happens for different values of `p` (percentage of points held out) and `rep` (number of repetitions of the experiment)?
- **III.D** For now we have been using the training set to obtain a classifier. Now we want to evaluate its performance by applying it to an independent test set.
  - Consider the test dataset `[Xte,Yte]` generated in I.D, and add noise to `Yte` as you did in III.B, to create `[Xte,Yte_noisy]`.
  - Take the best `k` you obtained by hold out cross validation in III.B, and use it to get a prediction from `Xtr,Ytrnoisy,Xte,` as you did in II.
  - Evaluate the prediction with respect to `Yte_noisy` (as you did in II.B2), and compare it to the validation error you had in III.B.

PART IV: What is a good dataset?

- **IV.A** Evaluate the results as the size of the training set grows `n`=10,20,50,100,300,... Keep in mind that `k` needs to be adjusted chosen accordingly.
- **IV.B** Generate more complex datasets with `MixGauss` function, for instance by choosing a larger variance during the data generation part.