# TP2: Face Recognition

pierre.perrault@inria.fr

Friday 9th November, 2018

**Abstract**

The report and the code are due in 2 weeks (deadline 23:59 28/11/2018). You will find instructions on how to submit the report on piazza, as well as the policies for scoring and late submissions. All the code related to the TD must be submitted.

A small preface (especially for those that will not be present at the TD). Some of the experiments that will be presented during this TD makes use of randomly generated datasets. Because of this, there is always the possibility that a single run will be not representative of the usual outcome. Randomness in the data is common in Machine Learning, and managing this randomness is important. Proper experimental setting calls for repeated experiments and confidence intervals. In this case, it will be sufficient to repeat each experiment multiple times and visually see if there is large variations (some experiments are designed exactly to show this variations).

Whenever the instructions require you to complete a file this means to open the corresponding file, and complete it according to the instructions all the following sections

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% use the build_similarity_graph function to build the graph W  %
% W: (n x n) dimensional matrix representing                     %
%    the adjacency matrix of the graph                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

YOUR CODE HERE

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

For each of these file the documentation at the top describes their parameters.

If you use Matlab, before running any code, move to the root directory where `load_td.m` is located, and run it to properly set the required Matlab paths and import packages.

# 1 Harmonic Function Solution

Semi-supervised learning (SSL) is a field of machine learning that studies learning from both labeled and unlabeled examples. This learning paradigm is extremely useful for solving real-world problems, where data is often abundant but the resources to label them are limited. We will use the following notation:

- $G = (V, E)$ is a weighted graph where $V = \{x_1, \ldots, x_n\}$ is the vertex set and $E$ is the edge set
- Associated with each edge $e_{ij} \in E$ is a weight $w_{ij}$. If there is no edge present between $x_i$ and $x_j$, $w_{ij} = 0$.
- Associated with each node there is a label with value $y_i \in \mathbb{R}$.
- Only a subset $S \subset V$, $|S| = l$ of the nodes' label is revealed to the learner, the remaining $u = n - l$ nodes are placed in the subset $T = V \backslash S$.

We wish to predict the values of the vertices in $T$. In doing so, we would like to exploit the structure of the graph. Since we believe that nodes close in the graph (similar) should share similar labels, we would like to have each node be surrounded by a majority of nodes with the same label. In particular, if our recovered labels are encoded in the vector $\mathbf{f} \in \mathbb{R}^n$ we can express this principle as

$$f_i = \frac{\sum_{i \sim j} f_j w_{ij}}{\sum_{i \sim j} w_{ij}}$$

where we use $f_i = f(x_i)$.

If we want to give a motivation for this belief, we can draw this interpretation in terms of random walks. In particular if the weight $w_{ij}$ expresses the tendency of moving from node $x_i$ to node $x_j$, then we can encode transition probabilities as $P(j|i) = \frac{w_{ij}}{\sum_k w_{ik}}$. If we compute a stationary distribution, it will be a valid solution for our criteria.

The same preference can be expressed with a penalization on all solutions that are not "smooth" w.r.t. the graph weights. This can be expressed as a function using the Laplacian matrix $L$

$$\Omega(\mathbf{f}) = \sum_{i \sim j} w_{ij}(f_i - f_j)^2 = \mathbf{f}^T L \mathbf{f}$$

Initially, we assume that the labels that we receive with the data are always correct, and it is in our best interest to enforce them exactly. In practice,

this means that we have first to guarantee that the labeled point will be correctly labeled, and then to promote smoothness on the unlabeled points. As an optimization problem, this can be formulated as

$$\min_{\mathbf{f}\in\{\pm 1\}^n} \infty \sum_{i=1}^{l} \left(f(x_i) - y_i\right)^2 + \lambda \sum_{i,j=1}^{n} w_{ij} \left(f(x_i) - f(x_j)\right)^2 .$$

If we relax the integer constraints to be real and fully enforce the label constraints, we end up with

$$\begin{aligned} &\min_{\mathbf{f}\in\mathbb{R}^n} \sum_{i,j=1}^{n} w_{ij} \left(f(x_i) - f(x_j)\right)^2 \\ &s.t. \quad y_i = f(x_i) \quad \forall i = 1, \ldots, l \end{aligned}$$

1.1. Complete `hard_hfs.m` and `two_moons_hfs.m` . Select uniformly at random 4 labels ($S$), and compute the labels for the unlabeled nodes ($T$) using the hard-HFS formula. Plot the resulting labeling and the accuracy.

1.2. At home, run `two_moons_hfs.m` using the `data_2moons_large.mat`, a dataset with 1000 samples. Continue to uniformly sample only 4 labels. What can go wrong?

What happens when the labels are noisy, or in other words when some of the samples are mislabeled? In some cases relabeling nodes might be beneficial. For this reason, soft-HFS seeks to strike a balance between smoothness and satisfying the labels in the training data. Define $C$ and $\mathbf{y}$ as

$$C_{ii} = \begin{cases} c_l & \text{for labeled examples} \\ c_u & \text{otherwise.} \end{cases} \qquad y_i = \begin{cases} \text{true label} & \text{for labeled examples} \\ 0 & \text{otherwise.} \end{cases}$$

Then soft-HFS's objective function is

$$\min_{\mathbf{f}\in\mathbb{R}^n} (\mathbf{f} - \mathbf{y})^\mathsf{T} C (\mathbf{f} - \mathbf{y}) + \mathbf{f}^\mathsf{T} L \mathbf{f}$$

1.3. Complete `soft_hfs.m` and test it with `two_moons_hfs.m` . Now complete `hard_vs_soft_hfs.m` . Compare the results you obtain with soft-HFS anc hard-HFS.

## 2 Face recognition with HFS

We can now start to think of applying HFS to the task of face recognition, or in other words to classify faces as belonging to different persons. Since faces all share common features, it is a effective idea to leverage a large quantity of unlabeled data to improve classification accuracy. As our first exercise, we will begin by completing the code necessary to define the similarity between faces. To extract the faces we will use OpenCV face detection software to localize them, and the same library to apply a series of preprocessing steps to improve their quality. Complete `offline_face_recognition.m` to classify the faces, and plot the results.

2.1. How did you manage to label more than two classes?

2.2. Which preprocessing steps (e.g. cv.GaussianBlur, cv.equalizeHist) did you apply to the faces before constructing the similarity graph? Which gave the best performance?

2.3. Does HFS reach good performances on this task?

Now try to augment the dataset with more unlabeled data, and test if additional data improves performance. In the archive `extended_dataset.tar.gz` you will find additional pictures to expand (augment) the dataset you already processed. The file `offline_face_recognition_augmunted.m` starts as an identical copy of `offline_face_recognition.m` modify it to handle your extended dataset, and answer the following questions

2.4. Did adding more data to the task improve performance? If so, which kind of additional data improves performance.

2.5. If the performance does not improve when adding additional data, try to justify why. Which kind of additional data degrades performance instead of improving it?