TP2: Online Face Recognition

daniele.calandriello@inria.fr

Tuesday 24th February, 2015

Abstract

The report and the code are due in 2 weeks (deadline 23:59 10/3/2015). You can send them by email to `daniele.calandriello@inria.fr`, with subject **TD2 Graphs In ML Name Surname**. Naming the attachments something along the lines of `TD2_Graphs_ML_report_name_surname.{pdf,doc}` and `TD2_Graphs_ML_code_name_surname.{zip,gzip}` would be greatly appreciated. All the code related to the TD must be submitted, to provide background for the code evaluation. All submission that will arrive late will be linearly penalized. The maximum score is 100 for all submissions before 23:59 10/3/2015 and 0 for all submissions after 23:59 13/3/2015 (e.g. the maximum score is about 70 at 21:30 11/3/2015). *Material on* `http://chercheurs.lille.inria.fr/~calandri/`

A small preface (especially for those that will not be present at the TD). All 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 huge variations (some experiments are designed exactly to show this variations).

Before running any code, move to the root directory where `load_td_path.m` is located, and run it to properly set the required Matlab paths. Remember to also add the path of `mexopencv`.

1 Semi Supervised Learning and 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. Consider a weighted graph $G = (V, E)$ 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 $x_j, w_{ij} = 0$. In other words, our edges encode the similarity of two adjacent nodes. Imagine a situation where a subset of these vertices are labeled with values $y_i \in \mathbb{R}$. We wish to predict the values of the rest of the vertices. In doing so, we would like to exploit the structure of the graph. In particular, in our approach we will assume that the weights are indications of the affinity of nodes with respect to each other and consequently are related to the potential similarity of the y values these nodes are likely to have.

Two concept can easily encode this preference in our solutions. Since we believe that similar nodes should share similar labels, we would like to have each node be surrounded by a majority of similarly labeled nodes. 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 $f_i = f(x_i)$.

This has a clear 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 $P(j|i) = \frac{w_{ij}}{\sum_k w_{ik}}$.

When looking into Spectral Clustering in the first TD, we saw that our goal was to find a solution that was smooth w.r.t. the graph weights. This concept can be be similarly encoded by enforcing smoothness in the label values according to the weights. In particular, if our recovered labels are encoded in the vector $\mathbf{f} \in \mathbb{R}^n$ we have

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

and from the previous TD we already know that

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

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

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

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

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

1.1. Complete `compute_hfs.m` . Collect 100 samples from the Two Moons udistributions with parameters `[1,0.2,0]`, select uniformly at random only 4 labels, and compute the labels for the unlabeled nodes using the constrained HFS (CHFS) formula. Plot the resulting labeling and the accuracy.

1.2. At home, raise the number of samples to 1000, and still uniformly sample only 4 labels. What can go wrong?

What happens when the labels are noisy, or in other words when some of the labels are wrong? In some cases relabeling nodes might be beneficial. For this reason, the unconstrained Harmonic Function Solution seeks to strike a balance between smoothness and satisfying the labels in the training data.

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

where C is defined as

$$C_{ii} = \begin{cases} c_l & \text{for labeled examples} \\ c_u & \text{otherwise.} \end{cases}$$

and $\mathbf{y}$ is a pseudo-targets with

$$y_i = \begin{cases} \text{true label} & \text{for labeled examples} \\ 0 & \text{otherwise.} \end{cases}$$

1.3. Modify `compute_hfs.m` to be able to find also the soft HFS (SHFS) solution. Test it on the Two Moons dataset with parameters `[1,0.2,0.1]` and 20 labels. Compare the results you obtain with SHFS anc CHFS.

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.

2.1. Complete `face_similarity_function.m` . To compute the similarity between nodes we will use an inverse exponential, but the distance will be measured using a weighted norm to take into account changes in lightning condition. Given an $n \times n$ image, each pixel must be premultiplied by the factor

$$w_{i,j} = 1 - \frac{1}{1.5n^2}\left((i-n)^2 + 0.5(j-n)^2\right)$$

2.2. Complete `offline_face_recognition.m` to classify the faces, and plot the results. How can you manage more than two classes?

3 Online SSL

Semi-Supervised Learning was introduced as a solution designed for problems where collecting large quantities of Supervised training data (usually labels) is not possible. On the other hand, in SSL scenarios it is usually inexpensive to obtain more samples coming from the same process that generated the labeled samples, but without a label attached. A simple approach is to collect as much additional data as possible, and then run our algorithm on all of it, in a batch or off-line manner. But in other cases it is possible to start the learning process as early as possible, and then refine the solution as the quantity of available information increases. An important example of this approach is stream processing. In this setting, a few labeled examples are provided in advance and set the initial bias of the system while unlabeled examples are gathered online and update the bias continuously. In the online setting, learning is viewed as a repeated game against a potentially adversarial nature. At each step t of this game, we observe an example x_t , and then predict its label f_t . The challenge of the game is that after the game started we do not observe the true label y_t. Thus, if we want to adapt to changes in the environment, we have to rely on indirect forms of feedback, such as the structure of data. Another difficulty posed by online learning is computational cost. In particular, when t becomes large, a naive approach to SHFS, such as recomputing the whole solution from scratch, has prohibitive computational costs. Especially in streaming settings where near real-time requirements are common, it is imperative to have an approach that has scalable time and memory costs. Because most operations related to SHFS scale with the number of nodes, one simple but effective approach to scale this algorithm is subsampling, or in other words to compute an approximate solution on a smaller subset of the data in a way that generalizes well to the whole dataset. Several techniques can give different guarantees for the approximation. As you saw in class, incremental k-centers [1] guarantees on the distortion introduced by the approximation allows us to provide theoretical guarantees.

Algorithm 1 Incremental k-centers

```
1: an unlabeled $\mathbf{x}_t$, a set of centroids $C_{t-1}$, $\mathbf{p}^c, \mathbf{p}^n, \mathbf{b}$
2: if $(|C_{t-1}| = k + 1)$ then
3:     Find two closest centroids $c_{add}$ that will forget the old centroid and will point
       to the new sample that just arrived, and $c_{rep}$ that will take care of representing
       all nodes that belonged to $c_{add}$
4:     If necessary $R \leftarrow mR$. If this happens, reset the taboos to only the labeled
       nodes.
5:     All $p_i^n$ that pointed to $c_{add}$ must be updated. To find the new node that they
       must point to, we use $\mathbf{p}^c$.
6:     We mark $c_{rep}$ as taboo, to avoid merging together too many close centroids.
7:     $c_{add}$ will now represent the new node, we must update accordingly $\mathbf{p}^c, \mathbf{p}^n$ and
       the stored centroids.
8: else
9:     $C_t \leftarrow C_{t-1}$
10:    $x_t$ is added as a new centroid $c_t$ and $\mathbf{p}^c, \mathbf{p}^n$ are updated accordingly
11: end if
```

There is a lot of practical consideration to keep in mind when implementing this kind of algorithms.

- The labeled nodes are fundamentally different from unlabeled ones. Because of this, it is always a good idea to keep them separate, and never merge them in a centroid. In the implementation this is accomplished with a taboo list $\mathbf{b}$ that keeps track of nodes that cannot be merged.
- Altought the doubling algorithm has this name, 2 is not always the optimal

constant to increase the R constant, neither theoretically or in implementation. A large R factor provides worse performance, so we should set it as low as possible as long as we can guarantee the invariants.

- Ammortized cost matters, especially in streaming application. It is not always possible to stop execution to repartition the centroids, and it is often preferrable to pay a small price at every step to keep execution smooth. In our case, the centroids are updated at every step.

The most important variables for the centroid update are the `centroids_to_nodes_map` and the `nodes_to_centroids_map`. They allow us to extract labels from the solutions, and to correctly mantain information on the multiplicities of every centroid. `centroids_to_nodes_map` $\mathbf{p}^c$ is a k dimensional vector such that p_i^c is the index of the nodes representative of the i-th centroid. On the other hand `nodes_to_centroids_map` $\mathbf{p}^n$ is the t dimensional vector that maps every node to the centroid that represents him. Whenever a new node arrives, and we have too many centroids, we choose the two closest centroids c_{add}, that will forget the old centroid and will point to the new sample that just arrived, and c_{rep}, that will take care of representing all nodes that belonged to c_{add}.

3.1. Complete `update_centroids.m` using the pseudocode from 1.

Computing the solutions from the quantized graph using SHFS is pretty straightforward.

Algorithm 2 Online HFS with Graph Quantization

```
1: $\mathbf{f}$ Input
2: Compute $L_t$ of $G(VWV)$
3: Infer labels
4: Predict $\widehat{y}_t = \mathrm{sgn}\left(\mathbf{f}_u\left(t\right)\right)$. With the preceding construction of the centroids,
   $x_t$ is always present in the reduced graph and does not share the centroid
   with any other node.
```

3.2. Complete `compute_solution.m` following the pseudocode from 2

3.3. Use `create_user_profile.m` to capture a training set of labeled data of your face and someone else. Read the help for `cv.VideoCapture` to understand how to open a stream. Run `online_face_recognition.m` either live or on a previously recorded video.

References

[1] Moses CHARIKAR, Chandra CHEKURI, Tomas FEDER, and Rajeev MOTWANI. Incremental clustering and dynamic information retrieval. *SIAM journal on computing*, 33(6):1417–1440, 2004.