Attached programs

The programs for this TP are included in the zip archive I sent, and are regular MATLAB/Octave programs (tested with only Octave v4, on Linux only).

1 Semi Supervised Learning and Harmonic Function Solution (SSL with HFS)

Question 1.1

Remark: See the code for more details.

See below Figure 1 for the required graph (showing the resulting labeling and the accuracy).

We chose a 6-nn graph, with \( \sigma^2 = 1 \) (useless for knn). If we are lucky in the 4 random samples, we get a perfect labeling (accuracy 1).

Question 1.2

For this second experiment, with the same distribution but a bigger sample sets, we got exactly the same results. With a 6-nn graph, the plot is “denser”, but for both Hard- and Soft-HFS we got accuracy = 1 (I did not include the same plot in the report, see the zip archive). The main difference is their time consumption, Soft-HFS appeared quicker (1.7 seconds against 2.4 seconds).

Shortly, with a huge dataset like this one, with only two classes, there is a not-so-small chance to only pick 4 labels that are the same (class 1 or class 2). And if all the samples have the same labels, the propagation cannot goes well, and the accuracy will be really bad.

So I guess, if we are unlucky, we can have a very bad accuracy. However, I have no idea if we will have in average a good accuracy or not.
Question 1.3

For the Soft-HFS, we had to chose the two values for $c_l$ and $c_u$. Nothing was said about them in class, neither in the slides. We have no heuristic at all to chose them.

It seems that $c_l$ must be “big” in comparison to $c_u$ (they represent the confidence w.r.t. labeled examples, resp. unlabeled ones). I chased $c_l = 0.96$ and $c_u = 0.04$, but we really have no idea if this is good or not.

With this choice, Soft-HFS seems to work as well as Hard-HFS, at least on some tries (the samples are random so the accuracy is also random), see Figure 2:

![Figure 2: One try of 4 random labels ($S$) and Soft-HFS on $E \setminus S$, with accuracy = 1.](image)

The second step was to compare Hard- and Soft-HFS on hard_vs_soft_hfs.m. This time we selected 20 samples and not just 4, but the performance are about the same for both method (for this specific choice of $c_u, c_l$ – it would have taken a lot of time to try “them all” and pick the best choice), see Figure 3:

![Figure 3: 20 samples, comparison Hard- vs Soft-HFS, with accuracy $\approx 0.9$.](image)

2 Face recognition with HFS

Question 2.1

We used the following piece of code to complete the face_similarity_function.m:

---

1 And a quick search on Internet about Soft-HFS gave nothing, for example [http://www.bkveton.com/docs/icml2011.pdf](http://www.bkveton.com/docs/icml2011.pdf) does not talk about the values of these constants.
tmp = ((1:n) - n) .* ((1:n) - n); % like a meshgrid
% then i is tmp' and j is tmp:
tmp2 = repmat(tmp', 1, n) + 0.5 * repmat(tmp, n, 1);
W = ones(n) - 1/(1.5 * n^2) * tmp2;

This implements the given formula (with \(n\) number of samples):
\[
w_{i,j} = 1 - \frac{1}{1.5n^2} \left((i - N) + 0.5(j - N)^2\right).
\]

**Question 2.2**

We then worked on `offline_face_recognition.m`, which uses the provided data (10 small pictures of the face of 10 quite different people). Q sample of 2 pictures for these 10 people was plotted and is included in the zip archive.

We filled the skeleton function which was provided. My initial implementation of `hard_hfs.m` did not use an \texttt{argmax}\footnote{\texttt{a [-, ind] = max(Y, [], 2)} in Octave}, so I inlined these five lines in the `offline_face_recognition.m`:

```
% build the laplacian
L = diag(sum(W, 2)) - W;

u_idx = setdiff(1:num_samples, l_idx);
Yl = oneHot(Y_l);
% Hard HFS formula from the slides
Yu = L(u_idx,u_idx) \ (W(u_idx,l_idx) * Yl) ;

% compute the labels assignment from the HFS solution
[~,label] = max(Yu, [], 2);
% Final labels from these labels
final_label = Y;
final_label(u_idx) = label;
```

In order to generalize from 2 class (like in part 1) to \(K = 10\) class (for these 10 people), we used the classical trick to encode a \(K\) classification problem with a target vector \(Y \in \{0,1\}^K\) (it’s a one hot encoding, as done for instance when using a multinomial distribution, see the \texttt{oneHot.m} function for implementation).

The results of the face classification for these 10 people are plotted bellow (Figure 4):

![Figure 4: Classification of the faces.](image)

On the left, we have the exact labels (each row as a constant color, ie. each person gets his 10 faces correctly labeled – obviously, it’s the true labels!). On the right, we represent similarly the indices, and we see that only half of the person got some of their face mis-classified. The obtained accuracy was about 88%, which is not bad but not excellent.
Question 2.3
It seems that HFS was quite efficient here (88% seems good to me!), but it could be better. I am not sure of what kind of accuracy we expected, so I have no idea if the result we got was really good or bad.

Sorry, I ran out of time to add more unlabeled data (and I do not even know how we could have done that, as the small database of faces we used was limited to the already-used 10 faces per person). This TP, as the first one, was very long.

3 Online SSL

Question 3.1
We worked here on online_ssl_update_centroids.m, using the pseudo-code given in Algorithm 1 (and from the slides).

It would have been hard to implement it from scratch, but filling the blanks and understanding the role of each variable/structure was alright. I think I managed to get it done.

The data-structure used for this algorithm seemed unnatural for Octave at first, but it appear to work well (and is probably numerically efficient, so no worries on that aspect).

Question 3.2
Then we focus on online_ssl_compute_solution.m, using the pseudo-code given in Algorithm 2 (and from the slides).

It is a not-so-simple workaround the Hard-HFS (as implemented in part 1). Computing $L_t$ and $G$ was quick (note that the dependance in $t$ of $L_t$ is only in the $V$ vector, not $\tilde{W}_q$). See code 5:

\begin{verbatim}
W_q = V * W * V;  % Normalized adjacency matrix from V
D_q = diag(sum(W_q, 2));
L = D_q - W_q;   % Quantized Laplacian
Q = L + gamma_g * eye(num_centroids);  % gamma_g regularization
\end{verbatim}

Figure 5: Computing quantized Laplacian and $\gamma_g$-regularized Laplacian.

We then compute the HARD HFS solution with a piece of code similar to the one used in part 1 (see Code 6).

We followed the template and used clear variable names and comments when needed, so I hope the implementation for this parts 3.1 and 3.2 would still be read and graded, even I no concrete results were obtained in subpart 3.1.

Question 3.3 – not concluded
For the final part of this TP, the main online_face_recognition.m file, I understood the general idea of what we wanted to do, and the whole experimental setting was clear.

But I ran out of time and had to conclude without completing this part. I had to leave on Friday 27th (November) for the weekend for a family issue, without being able to complete the final experiment.

I have not tried to test my implementation of the two required functions from 3.1 and 3.2, but I really think they are correct (after checking from the pseudo-code several times).
all_idx = 1:num_centroids;
l_idx = all_idx(Y_mapped ~= 0);
u_idx = all_idx(Y_mapped == 0);
y = [Y_mapped(Y_mapped ~= 0) == -1, Y_mapped(Y_mapped == 0) == 1];

% compute the hfs solution
f_l = y;
Q_uu = Q(u_idx, u_idx);  % Regularized Laplacian L
W_ul = W_q(u_idx, l_idx);  % Sub quantized matrix G
f_u = Q_uu \ (W_ul * f_l);  % Solve system

% compute the labels assignment from the hfs solution
label = y;
y(sublabs == 0)
[~, label_1_2] = max(f_u, [ ], 2);  % 2*1-3 = -1, 2*2-3 = +1
label(u_idx) = 2*label_1_2 - 3;  % -> Only +1 -1 labels

Figure 6: Computing the labels $\in \{-1, 1\}$ with Hard-HFS on $\gamma_g$-regularized Laplacian.

Conclusion

**Remark:** I again found the TP to be too long at the end.
The aim of the TP was an amazing application, so I am sad to not have been able to reach a working prototype conclusion, mainly because of bad time management.

**Note:** I edited the images with [Gimp](https://en.wikipedia.org/wiki/GIMP) in order to remove useless white parts, but nothing was modified.