TP2: Semi-Supervised Learning (SSL)

Bastien Dechamps

November 26, 2019

1 Harmonic Function Solution (HSF)

1.1. Complete `hard_hfs` and `two_moons_hfs` . 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.

Figure 1 shows an example of the resulting labelling for the hard HFS solution, with $\sigma^2 = 1, k = 7, \gamma_g = 10^{-3}$ and a random walk laplacian normalization. It scored an accuracy of 1.

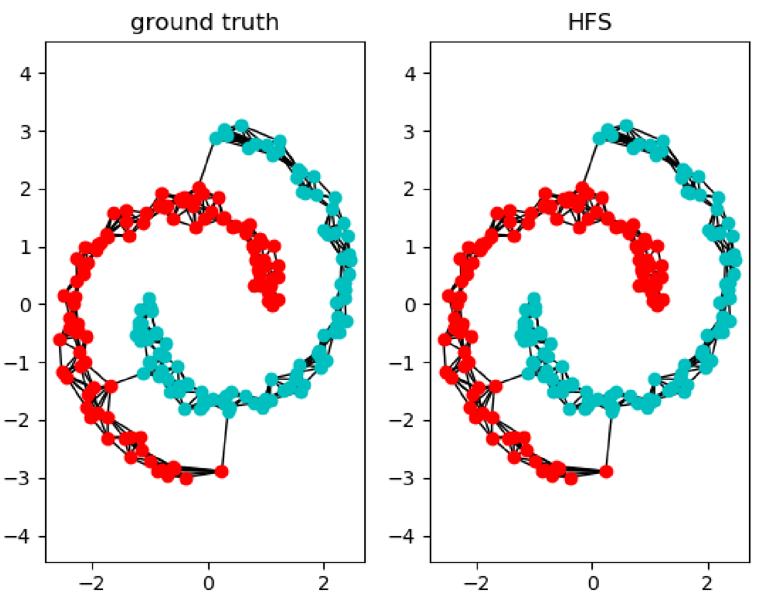


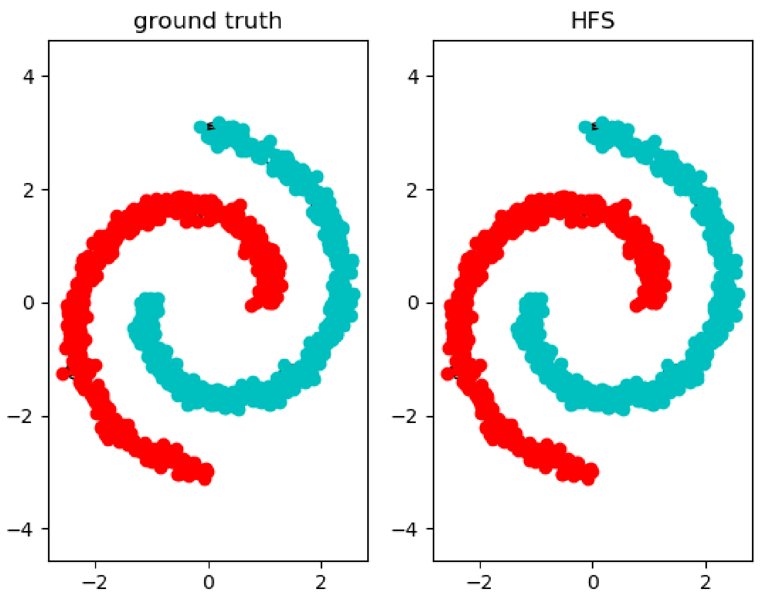
Figure 1: Hard HFS solution for a k-nn graph ($k = 7$)

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

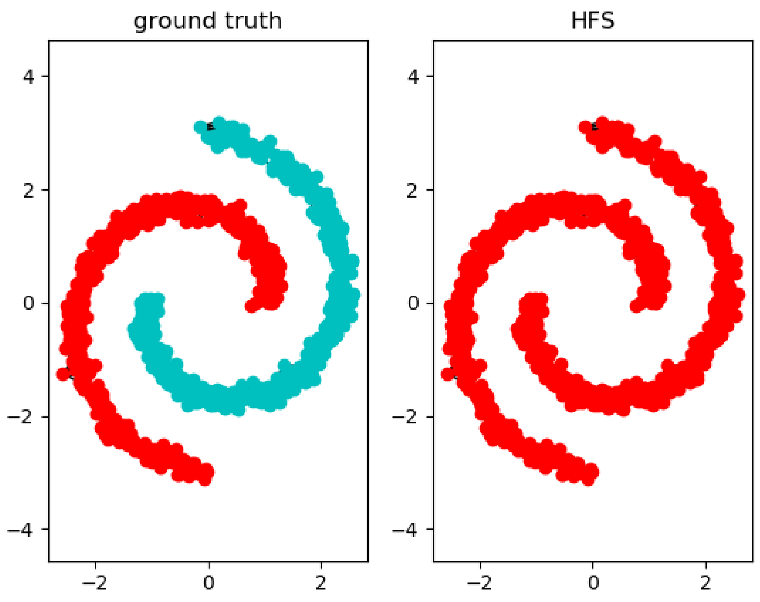
In figure 2a, we see that with enough labelled nodes, we find a perfect labelling of the two large moons. However, if all the labelled nodes are sampled from only one of the two moons (figure 2b), then, we can't propagate the other label to the other moon, and the resulting labelling is wrong.

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

Figure 3 shows the resulting labelling for the soft HFS solution with the same parameters than in question 1.1 and $c_l = 10^{-3}, c_u = 10^{-2}$.



(a) Good sampling



(b) Bad sampling

Figure 2: Hard HFS solution for `data_2moons_large.mat`

In figure 4, we represent the result of `hard_vs_soft_hfs` for the same parameters than above, with 4 mislabelled points among 20. We obtain an accuracy of 0.81 for the hard HFS and 1 for the soft HFS. Indeed, the soft HFS allows labelled nodes to be corrected if they are "surrounded" by too many nodes with another label. In the hard HFS, labeled nodes cannot be changed and propagate the error to the neighboring nodes.

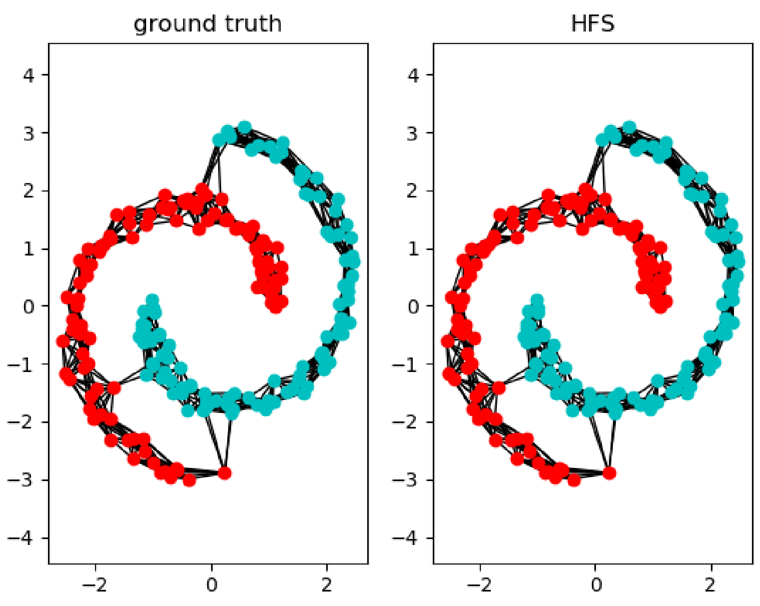


Figure 3: Soft HFS solution for a k-nn graph ($k = 7$)

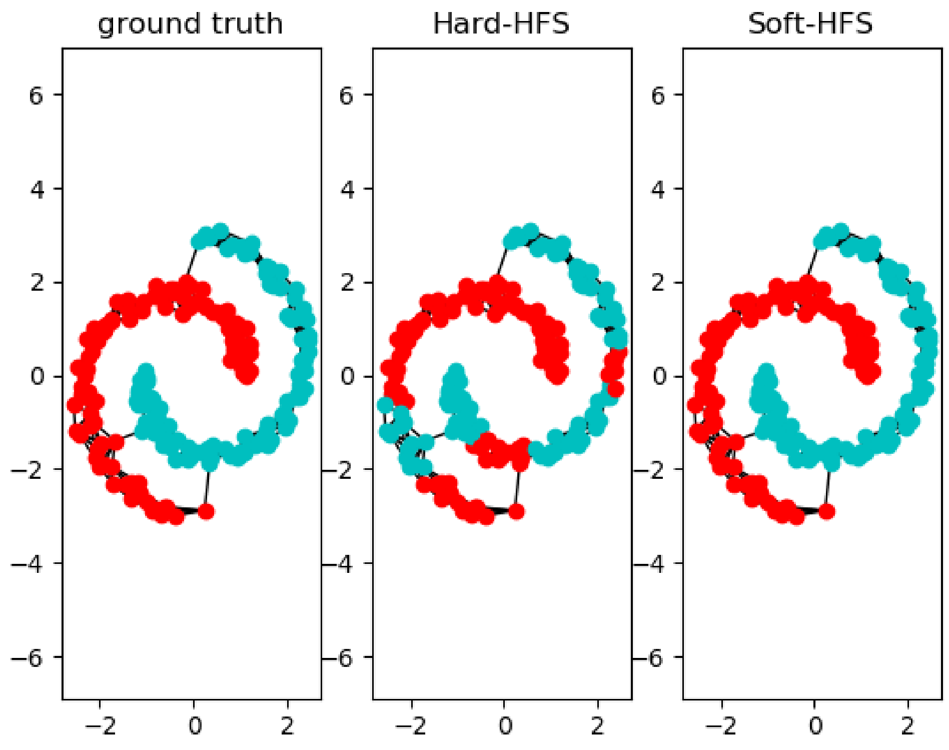


Figure 4: Hard vs soft HFS solutions for a k-nn graph ($k = 7$)

2 Face recognition with HFS

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

In the code, I encoded the labels with one hot vectors in $\{0,1\}^{n \times K}$ where K is the number of classes, rather than with vectors of labels in $\{1, ..., K\}^n$.

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?

We can apply several preprocessing steps such as brightness enhancing, histogram equalization or other filters such as `cv.boxFilter`,`cv.GaussianBlur`,`cv.bilinearFilter`. Brightness enhancing with histogram equalization gave the best accuracy (86%). The classification results are represented on figure 5.

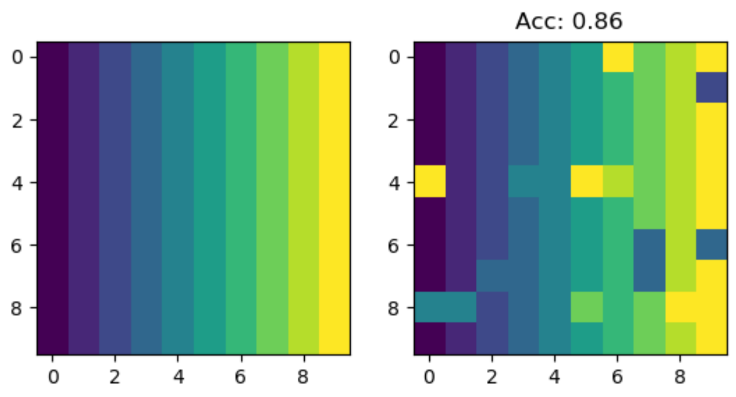


Figure 5: HFS solution for face detection

2.3. Does HFS reach good performances on this task?

Hard HFS and soft HFS perform the same overall on this dataset, with a slightly best accuracy for hard HFS in general (see figure 6).

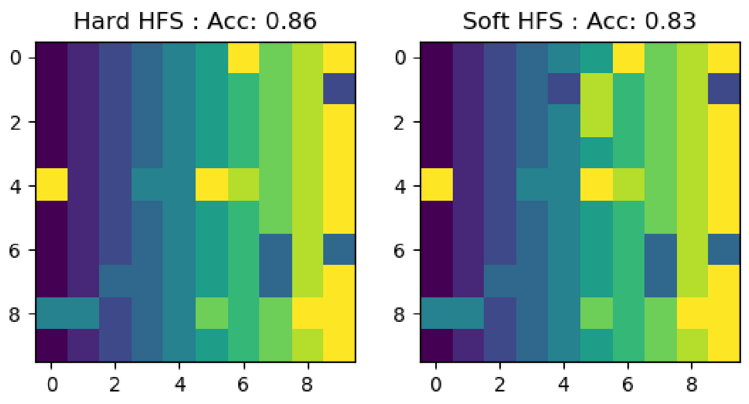


Figure 6: Hard vs soft HFS for face detection

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

I found that adding more data reduces considerably the accuracy. As shown on figure 7, the best score I could get by refining all the parameters is around 0.64. I applied the same preprocessing step than above, built a k-nn graph with $k = 7$ and applied soft HFS (the hard HFS gives very low accuracy in this case).

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

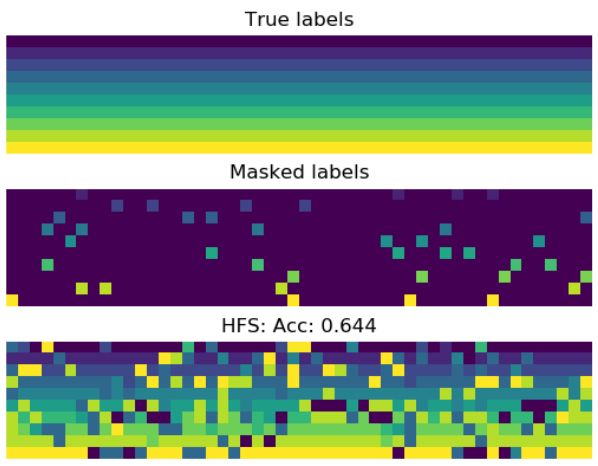


Figure 7: Soft HFS on augmented faces dataset

First, we still have 4 labelled faces for each person but now for many unlabelled faces, whereas in the small dataset we only had 10 faces per person. This may explain partly the loss of performance in this case. However, the soft HFS should work also with a few unlabelled nodes, so another reason may be more variations and noise in the faces (motion blur, rotation of the face, ...).

3 Online SSL

3.3. Read the function `preprocess_face` (in `helper_online_ssl.py`) and run `online_face_recognition`. Include some of the resulting frames (not too similar) in the report showing faces correctly being labeled as opposite, and comment on the choices you made during the implementation.

On figures 8 and 9, I displayed some frames I took with my roommates Mathieu and Caroline, each person having a user profile created. On some of her labelled photos, Caroline had glasses, but the algorithm managed to recognize her anyway. I used a k-nn graph with $k = 10$, $\gamma_g = 10^{-3}$, and a random walk normalization to have scores between 0 and 1 so that they can be interpreted as probabilities.

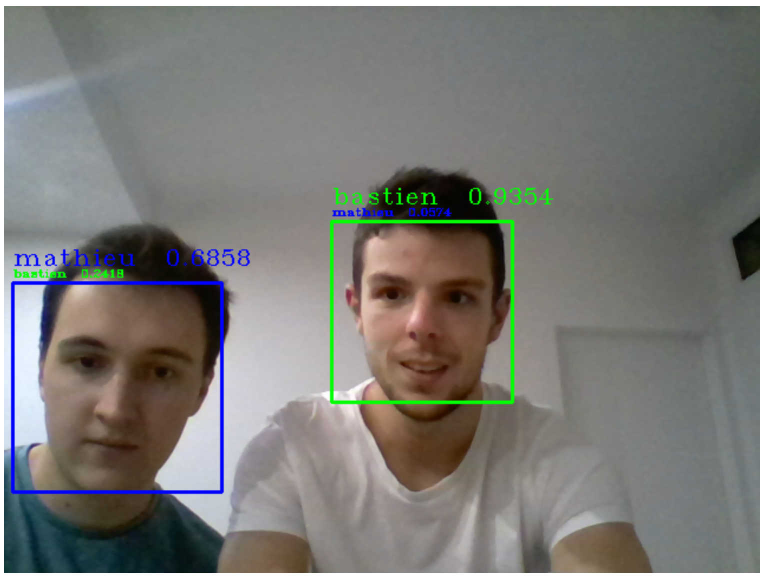


Figure 8: Me (Bastien) and Mathieu

3.4. What happens if an unknown person's face is captured by the camera? Modify your code to be able to disregard faces it cannot recognize, and include some of the resulting frames (not too similar) in the report showing unknown faces correctly being labeled as unknown.

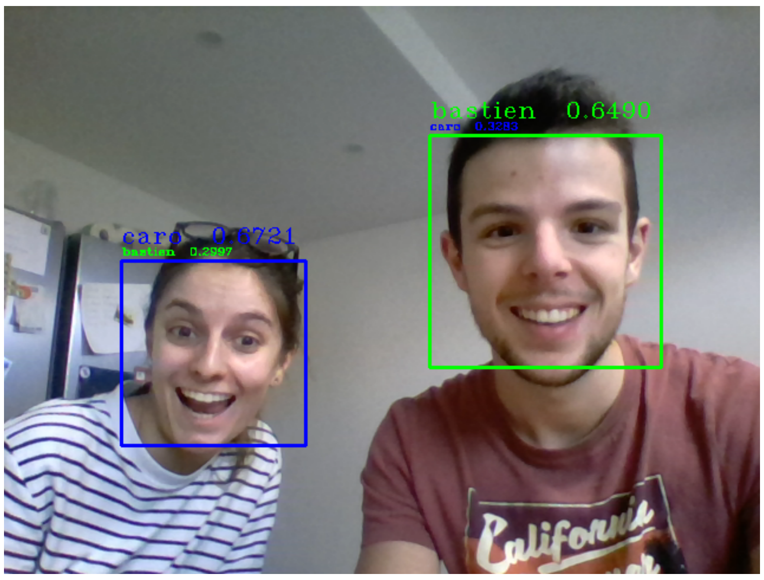


Figure 9: Me (Bastien) and Caroline

If an unknow person's face is captured, it is labeled as one of the user profiles (with the most similar face theorically), but with a low score. We can add a threshold condition on the score (given by the online HFS solution) below which the face is considered as unknown. On figure 10 (resp. figure 11), Mathieu's profile (resp. Caroline's profile) is not created, and he (resp. she) appears as unknown.

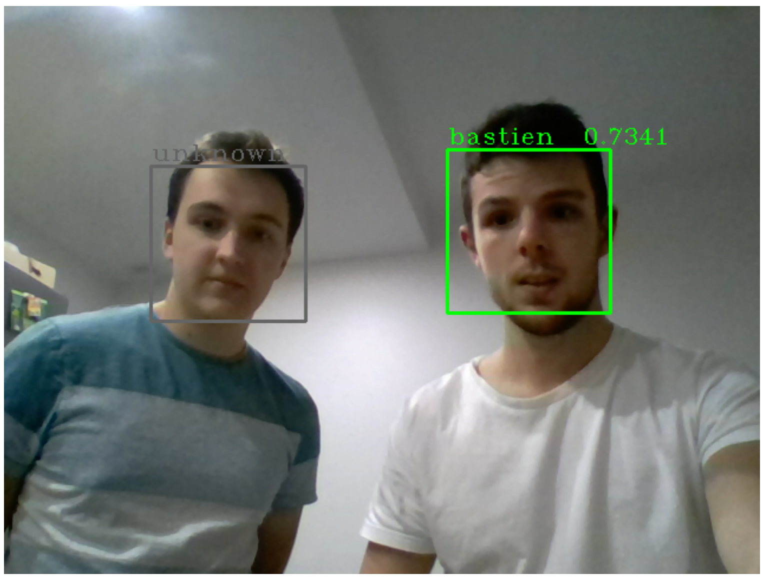


Figure 10: Me (Bastien) and Mathieu (unknown)

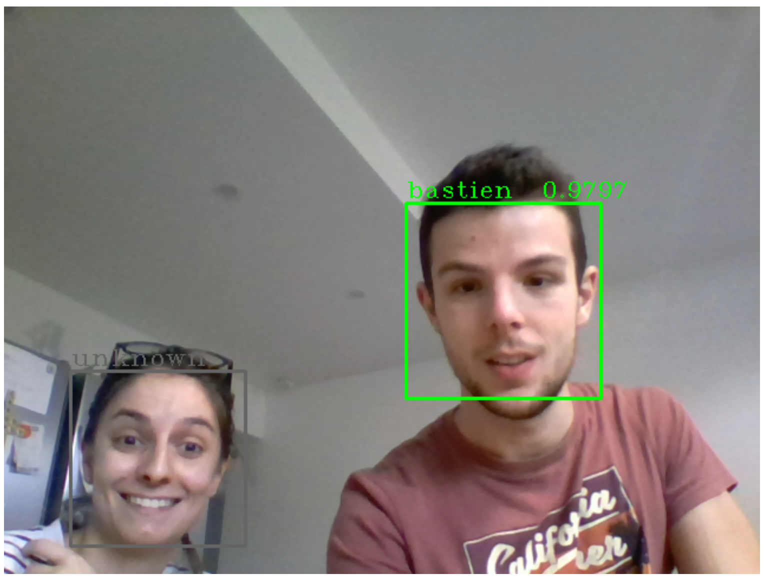


Figure 11: Me (Bastien) and Caroline (unknown)