Unsupervised Learning - Practical session IMA205

Pietro Gori

30 January 2019

PLEASE NOTE: you should have all toolboxes of Matlab installed. If you are using the computers of Telecom, please use Matlab 2018. Open a terminal, type matlab-2018, Enter.

This Practical session is about unsupervised learning. We will use the dimensionality reduction and clustering techniques presented this morning to analyze toy examples, recognize faces and segment skin lesion images.

You have one week (06/02) to update a small report (code + answers) to the *site pédagogique* of IMA205 under the section *Reports-TP*. You can answer in French or English. The deadline is 23:59 of the 06th of February. I remind you that the report is mandatory and evaluated. **All reports uploaded after the deadline will not be evaluated, namely grade equal to 0**.

1 Data

There are three main functions, a folder with the data and another folder with Matlab functions. The three main functions are about the three topics of this practical session.

2 Toy examples

Use the main function *Toy_main.m.*

Here you can play with the toy examples shown during the lecture. Try to understand the functions, change the parameters and comment. (Hint: comment means trying to build examples that confirm what seen this morning...do not spend too much time on this).

3 Face recognition

Use the main function *Face_main.m.*

Load the original images present in the files *'YaleB_32x32.mat'*. This is a small part of the freely available Extended Yale Face Database B downloaded from `http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`. It contains 2414 cropped images resized to 32x32 pixels. Every image is represented as a vector 1x1024 and all images are stacked in a matrix called data. There are 38 subjects with around 64 near frontal images per individual under different illumination conditions. Once loaded and normalised the data, such that the pixels are between 0 and 1, you can plot some images using the function *'imshow'*.

3.1 Goal

The goal of this part is to evaluate the performance of the dimensionality reduction techniques presented this morning for face recognition. We divide the data-set into two parts, training and test, using the random indexes present in *'Random_partition.mat'*. For every dimensionality reduction technique, you will first extract a set of basis images from your training data-set. Then, you will project the test subjects in this new basis and use the nearest neighbor algorithm to evaluate the performance of the dimensionality reduction technique.

3.2 PCA

We will first use the *'pca_lecture.m'* function with the training data. It computes the scores, eigenvectors and eigenvalues. The eigenvectors represent the basis images and they are usually called *'Eigenfaces'*. We project both training and test data onto the eigenvectors that explain 95% of the variability of the training set. We thus obtain two vectors of scores which are then used for evaluating the performance of the algorithm.

Look first at the *'pca_lecture.m'* function, try to understand it and answer the following questions:

1. Why do we need to center the data before computing a PCA ? If you want, you can use the previous toy examples to answer this question.
2. Let x_p and x_q be two row-vectors representing two images, U an *orthogonal* matrix whose columns are the eigenvectors of X and $y_p = x_p U$, $y_q = x_q U$, check that $x_p x_q^T = y_p y_q^T$. This shows that $Y = XU$ is a linear transformation that preserves inner products.
3. Let X be the original data, a matrix $[N, d]$, and Y the scores of a PCA keeping all eigenvectors, which means that Y is also a matrix $[N, d]$. Are X and Y equal ? If not, why ? What would you use (generally speaking) in a machine learning problem ? Why ?
4. Let C be the covariance matrix of X and $C = UDU^T$ its eigen decomposition. Show that the covariance matrix of $Y = XU$ is D.
5. Plot the eigenfaces. Do the they seem “real” ?

Now it's time to evaluate the performance of PCA using the nearest neighbor algorithm. Look at *'nearest_neighbor.m'*. For every test image, we first compute the angle in radiants between its score and all the scores of the training images. Then, we assign the test image to the subject of the training image with the lowest angle.

1. How are the angles computed ?

3.3 KPCA

In this section, we are going to do exactly the same procedure as before but using Kernel-PCA with a Gaussian kernel. Open the function *'Kpca_gaussian_lecture.m'* and try to understand the code. Remember that we need to center the kernel matrix $[\tilde{\mathbf{K}}]_{ij} = \langle \phi(x_i) - \frac{1}{N}\sum_{s=1}^{N} \phi(x_s), \phi(x_j) - \frac{1}{N}\sum_{s=1}^{N} \phi(x_s) \rangle$ and that, once computed the basis vectors in the training set $\{\alpha_i\}$, we can compute the score for a test sample t using the following equation:

$$y_i(t) = \sum_{j=1}^{N} a_{ij} \langle \phi(t) - \frac{1}{N}\sum_{s=1}^{N} \phi(x_s), \phi(x_j) - \frac{1}{N}\sum_{s=1}^{N} \phi(x_s) \rangle = \sum_{j=1}^{N} a_{ij} \tilde{k}(t, x_j) \qquad (1)$$

Answer these questions:

1. Why the basis vectors $\{\alpha_i\}$ are not plotted as in PCA ?
2. Create a new function *'Kpca_poly_lecture.m'* where you change the kernel to $k(x, y) = \langle x, y \rangle^d$. Evaluate the performance of this new kernel.

3.4 Performance

1. Compare performance, computational time and "interpretability" of the results. Which one is the best in your opinion ? Why ?
2. Is it worth it in your opinion to compute PCA and KPCA ? Why not using the original pixel intensities ?

3.5 ICA and NNMF - ! Optional !

For those of you who want to know more about this domain, I have also put the results for Independent Component Analysis (ICA) and Non-negative Matrix Factorization (NNMF).

3.5.1 ICA

The fast ICA algorithm is implemented in *'fastICA_ lecture.m'*. Every image x_i can be seen as a linear combination of basis images: $x_i = \sum_j c_{ij} b_j$ where c_{ij} is a coefficient and b_j is a basis image. ICA can be used in two different ways for face recognition. We can look for a set of statistically independent basis images b_j (as in the slides of the lecture, not shown here) or for a set of statistically independent coefficients c_{ij} (shown here).

In the second architecture, we compute $X = AS$, where every column of X is an image and rows are pixels. In this case, we consider the pixels as observations and we look for a set of statistically independent coefficients contained in the rows of S and a set of basis images in the columns of A.

Instead than using the original training data X as input matrix, we are going to use the scores $Y = XL$, computed with a linear PCA, to reduce the computational time. If you want, you can of course use the original data but it will take much more time to converge.

More precisely, we will use Y^T as input matrix since we want to have images in the columns. It results, $AS = Y^T = L^T X^T$ which entails $X^T = LW^T S$ (since $A = W^T$). The columns of LW^T contain the basis images whereas the columns of S contain the statistically independent coefficients used to test the performance of the algorithm. For the test set we compute $S_{test} = W_{train} Y_{test}^T$.

4 Skin lesion segmentation

Use the main function *Dermo_ main.m.*

In this section, you will use the algorithm of k-means to segment skin lesion images. In the folder *'Data/Images_ dermo'* you'll find three images of the Challenge with the ground truth segmentations. You can show images and segmentations by simply modifying the *imName.*

4.1 Goal

The goal of this section is to delineate the contours (i.e. segment) of the skin lesions using k-means. Try first to change the maximum number of classes K and analyze the results.

1. How many classes should you look for ? Is K the same for all images ? Why ?
2. Choose one of the classes as segmentation mask for the skin lesion and compare it with the ground truth segmentation using the Dice/Jaccardi index (matlab function *dice/jaccardi*).
3. Propose a way to automatically select the class with the segmentation mask (no need to implement it, just explain how you would do).

4. Show that $\sum_{i=1}^{N} z_{ik}(x_i - \mu_k)^2 = \frac{1}{2N_k} \sum_{i=1}^{N} \sum_{\substack{i=1 \\ j \neq i}}^{N} z_{ik} z_{jk} (x_i - x_j)^2$
5. In K-means we recompute the average at each iteration. The average is not constrained to be one of the original observations. It is usually an *interpolation* of the original observations. How would you change the Lloyd's algorithm to constrain the average to always be one of the original observations ?
6. (Optional) Try to code the Lloyd's algorithm and compare it with the version of Matlab.
7. (Optional) Implement the Lloyd's algorithm using the Kernel trick.