67750: Advanced Practical Machine Learning 2017/8

Exercise 2 - Unsupervised Image Denoising

Prof. Yair Weiss *TA: Daniel Gissin*

1 Theoretical Questions

Note that this section is not graded and you do not have to hand it in.

1.1 MLE Calculation

In class we calculated the MLE of a sample drawn from a Bernoulli distribution and a Gaussian distribution.

You are now the manager of the Rothberg cafeteria, and you collect N samples of the time it took from the moment person i asked for a salad until he got his salad. You assume, as one does, that the service time is distributed exponentially ($p(x) = \lambda e^{-\lambda x}$ for $x \geq 0$). Calculate the parameter λ using Maximum Likelihood Estimation.

1.2 MLE in the EM algorithm

In class we saw that the Expectiation of the log likelihood of the Gaussian Mixture Model can be written as:

$$\mathbb{E}[LL(S, Z, \theta)] = const + \sum_{i=1}^{n} \sum_{y=1}^{k} c_{i,y} log(\pi_y N(x_i; \mu_y, \Sigma_y))$$

We then were able to maximize the above expectation w.r.t. the parameters of the GMM. Show that the MLE of the multinomial distribution of our latent variable is:

$$\pi_y = \frac{1}{n} \sum_{i=1}^{n} c_{i,y}$$

1.3 Multivariate Gaussian Optimal Estimator

We denote the clean signal by x, and the noisy signal by y. The relationship between them is given by:

$$y = x + \eta$$

where we assume η is Gaussian with zero mean and covariance matrix $\sigma^2 I$. For this exercise we will assume that σ is also known. As we discussed in class, this means that $y|x$ is Gaussian with mean x and covariance $\sigma^2 I$. This means that after we learn $p(x)$ we can form the joint probability $p(x, y)$ and use this to calculate the optimal estimate of x given y which we proved in class is given by $E(x|y)$.

In class we showed an analytical form for $E(x|y)$ when $p(x)$ is a scalar Gaussian:

$$x^* = \frac{\frac{1}{\sigma_x^2}\mu_x + \frac{1}{\sigma^2}y}{\frac{1}{\sigma_x^2} + \frac{1}{\sigma^2}}$$

Now let's assume that $p(x)$ is a **multivariate** Gaussian with mean μ and covariance Σ. Show that the optimal estimate of x given y is given by

$$x^* = \left(\Sigma^{-1} + \frac{1}{\sigma^2}I\right)^{-1}\left(\Sigma^{-1}\mu + \frac{1}{\sigma^2}y\right)$$

You may use the trick we saw in class for Gaussian distributions: $\mathbb{E}[x|y] = argmax_x(p(x|y)) = argmax_x(p(x,y))$

2 Practical Exercise

Please submit a single tar file named "ex2_<YOUR_ID>.tar.gz". This file should contain your code, along with an "Answers.pdf" file in which you should write your answers and provide all figures/data to support your answers (write your ID in there as well, just in case). Your code will be checked manually so please write readable, well documented code.

The exercises in our course are written with Python3 in mind, but if you wish to write your code in another language, please contact the TA.

If you have any constructive remarks regarding the exercise (e.g. question X wasn't clear enough, we lacked the theoretical background to complete question Y...) we'll be happy to read them in your Answers file.

2.1 Model Introduction

In class we learned the full algorithm for non-blind image denoising, under the assumption that $x \sim GMM$:

- For learning $p(x)$, we use the EM algorithm to estimate $\{\pi_y, \mu_y, \Sigma_y\}_{y=1}^k$.
- For denoising, given a noisy image y, we use a weighted average of k different Wiener filters and calculate the optimal estimate of the clean image x:

$$Weiner(y) = \left(\Sigma^{-1} + \frac{1}{\sigma^2}I\right)^{-1}\left(\Sigma^{-1}\mu + \frac{1}{\sigma^2}y\right)$$

$$x^* = \sum_{i=1}^{k} c_i Weiner_i(y)$$

  c_i here is the posterior probability that our image patch came from Gaussian i. This posterior probability is calculated in a similar way to how we calculate the $c_{i,y}$'s in the EM algorithm,

it is $Pr(Gaussian = i|y)$ where y is our noisy image patch. That can be calculated using Bayes' rule:

$$c_i = \frac{Pr(Gaussian = i, y)}{\sum_{j=1}^{k} Pr(Gaussian = j, y)} = \frac{Pr(Gaussian = i)Pr(y|Gaussian = i)}{\sum_{j=1}^{k} Pr(Gaussian = j)Pr(y|Gaussian = j)} \rightarrow$$

$$c_i = \frac{\pi_i N(y; \mu_i, \Sigma_i + \sigma^2 I)}{\sum_{j=1}^{k} \pi_j N(y; \mu_j, \Sigma_j + \sigma^2 I)}$$

Next, we will introduce two new models which extend the Gaussian Mixture Model, which you will also implement in this exercise (and learn using EM).

2.1.1 Gaussian Scale Mixture (GSM)

Here, instead of assuming that the images were samples from a mixture of any k Gaussians, we will assume that the k Gaussians all share the same covariance matrix, and the only thing that is different is the scale by which the covariance is multiplied. In other words, we will assume our patches are first sampled from a Gaussian with distribution $N(0, \Sigma)$, and then a coin is flipped to determine by which of the k possible scalars will x be multiplied:

$$i \sim Multinomial(\pi)$$

$$Z \sim N(0, \Sigma)$$

$$x = r_i Z \rightarrow x \sim N(0, r_i^2 \Sigma)$$

This model is less expressive than the normal Gaussian Mixture Model, but what we lose in expressivity we gain in making the convergence of EM to the optimal solution more likely. There are also other justifications for this model which we will not go into, that stem from the statistics of natural images.

In this model, the parameters that we need to learn are the parameters Σ, π and the k possible scalars $\{r_i\}_{i=1}^{k}$. To make things simpler for us, we will fix Σ to be the sample covariance matrix of our data (like we do in the single multivariate Gaussian MLE calculation). The rest of the parameters will be learned using the EM algorithm, where the only new parameters that you haven't seen before are the scalars $\{r_i\}_{i=1}^{k}$. Those can be learned in the M-step of the EM algorithm using the following update (also derived using MLE):

$$r_y^2 = \frac{\sum_{i=1}^{n} c_{i,y} \cdot x_i^T \Sigma^{-1} x_i}{d \cdot \sum_{i=1}^{n} c_{i,y}}$$

Here $c_{i,y}$ is like we saw in class, the probability that x_i came from Gaussian y. d is simply the dimension of the Gaussian (i.e. 64 if we're dealing with $8x8$ patches).

Denoising for this model is identical to denoising for GMM, where now the covarince matrices will be $\{r_i^2 \Sigma\}_{i=1}^{k}$.

2.1.2 Independent Component Analysis (ICA)

ICA is a general model which is used in many fields, where we want to seperate a multivariate variable into independent subcomponents. this is done by assuming each of the variables is created using a different linear combination of independent (non Gaussian) variables. For our purposes, we will assume that there are d independent scalar random variables $\{s_i\}_{i=1}^{d}$, each distributed using a scalar GMM of k Gaussians ($k > 1$). We then assume our image patch is sampled in the following way:

$$x = As$$

This means that we first sample the d s's (each from a different mixture of k Gaussians), and then we take the vector s and multiply it by some square matrix A to get our image patch x (also d dimensional).

We won't show this explicitely, but in order to assure that our s's are independent (or at least uncorrelated), we can choose A to be the following matrix:

$$\Sigma = P\Lambda P^T$$

$$A = P$$

So simply diagonalizing our sample covariance can give us our A.

Now that we've made these assumptions, we can simplify our problem by denoising in the s-domain. Since $s = A^{-1}x = P^T x$, we can look at any image in the s domain by multiplying it by P^T.

Assuming we know the parameters of the d mixtures of k univariate Gaussians, we can take any noisy image y perform the following denoising procedure:

1. Move y to the s domain: $s_{noisy} = P^T y$.
2. Denoise each of the coordinates of s_{noisy} seperately using the Wiener filter to get $s_{denoised}$.
3. Move back to the image patch domain to get our denoised image: $x_{denoised} = P \cdot s_{denoised}$.

To learn the parameters of the d mixtures of k univariate Gaussians, we perform the following learning procedure:

1. Calculate A from our training set by estimating the covariance matrix.
2. Transform our training set to the s domain: $s_i = P^T x_i$
3. Learn each of the d coordinates in the s domain seperately using the EM algorithm, assuming each coordinate is sampled from a mixture of k univariate Gaussians.

2.2 Working with Image Patches

In order to avoid having to invert huge matrices, we will work with small image patches in this exercise (instead of full images). This means that we will assume that each image was created by stitching together $d \times d$ sized image patches (d will usually be 8) which were sampled from one the distribution of one of our models.

The actual procedure of building an image patch data set from full-sized images, along with the procedure of denoising a full-sized image using a model of image patches, is supplied in the supplementary code. You may go over the code if you want to learn how this can be done, but in general the idea is that every pixel in the noisy image is denoised by denoising the image patch around it and returning the single pixel's value after the full patch was denoised.

2.3 Exercise Requirements

The three models that you will need to implement in this exercise are the single multivariate Gaussian model (MVN), the GSM model and the ICA model. You will begin by implementing functions for learning the three models from the original training data. Then, you will implement functions for testing how the training set is explained by your model with log likelihood calculation functions. Finally, you will test your models by denoising images and checking how effective your models were. The overall requirements will be:

1. Write a function for learning each model using the EM algorithm (45 points)
2. Write a function for evaluating the log likelihood for each model (15 points)
3. Write a function for denoising pictures for each model (15 points).
4. Provide a comparison of these three models in terms of run time and of the log-likelihood of the clean test set. Provide plots which demonstrate your EM implementation works (improves the likelihood with each iteration and converges). Also, write which model you think is best and why. Provide qualitative evidence to support your conclusions (i.e. de/noised images). (25 points)
5. Provide a basic main function which demonstrates your implementation.

The supplimentary code, along with the function you are required to implement, can be found in the "Image Denoising.py" file.

2.4 Data

A pickle data set of natural images can be found along with the supplimentary code (seperate data set for training and testing) - you can either use this dataset, or any other collection of images you wish. In any event, be sure to transform the images to grayscale, rescale values to $[0, 1]$, and remove the mean of the dataset (you may use the supplied function that does that). There is also a function called "images_example" which demonstrates how to load the data and plot it.

2.5 Supplementary Code & Function Headers

This exercise comes with some helper functions that should make handling the images a little bit easier, so you can focus on the actual machine learning task. You may use the supplied code as it is or change it as you see fit (just make sure to let us know in your "Answers.PDF" file if you changed anything). The supplied code allows you to standardize your images (using the

"greyscale_and_standardize" function) and sample patches from a given image (using the "sample_patches" function). You can look at the "images_example" function to see how you can load the data so that it's ready for denoising (the "sample_patches" function also does the standardizing.

We also provide you with a function for denoising an image given pointers to your denoising function and model (using the "denoise_image" function) and a basic function for testing your implementation ("test_denoising"), but you are encouraged to test your code in other ways (for example, by creating synthetic data and seeing if your EM implementation learn the original parameters). You may look at the function headers to learn more about these supplementary functions.

Finally, we have also supplied you with different basically implemented classes for the different models and for a general GMM model, to direct you towards an object oriented implementation (since you can implement a single generic EM algorithm for GMM models and use it along with simple wrappers for the three models). That being said, you may implement this exercise in any way you want (you can change any function signature you want, add functions, delete functions and so on), just as long as your code is readable and clear (and works).

2.6 Elaboration And Advice

2.6.1 Log Space

As discussed in class, since our image patches are high dimensional, you may experience numerical instabilities when calculating the likelihood of patches. To deal with these issues, it would be wise to work in log space, saving each likelihood in its log likelihood form. Scipy has a few functions that can help with these kinds of calculation, like "logsumexp" which accepts arrays in log space and calculates $log(\sum_i exp(x_i))$ (basically a sum function that keeps things in log space). You can see an example of using this kind of function in the supplied function "normalize_log_likelihood", which normalizes a matrix in log space in a specified axis. You may use this function in your implementation as well.

Once you finish your calculations in log-space and you need to access the actual probabilities, just take the exponent...

2.6.2 Efficient Coding

As in most machine learning tasks, there is quite a bit of matrix and vector manipulation in this exercise. Try and avoid loops as much as possible, and use numpy functions as much as you can.

Still, your run time might not be too fast, especially in the ICA model which has to run many EMs in order to learn the parameters. We suggest pickling your models during training so you can easilly test your trained models without having to keep retraining them all the time.

2.7 Feedback

We would be happy to hear constructive comments about this exercise in your Answers.pdf file. Specifically:

1. How long did it take to complete the exercise?
2. Were there parts where you felt that you were working too hard on something that didn't teach you enough?
3. Did you feel that you lacked background for any of the questions?