University of BRISTOL

Inference

Machine Learning (COMS30007)

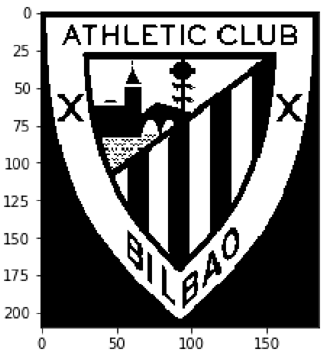
Bittor Alana
lo18144 - 97016

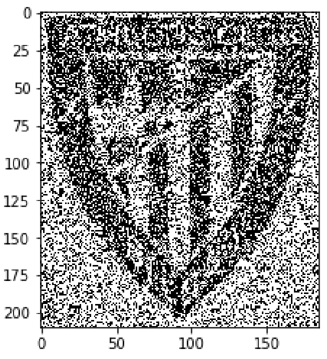
Jack Morgan
jm15357 - 27255

Alessandro Scibetta
as16155 - 38352

October 16, 2019

Question 1


(a) Original binary image chosen for analysis


(b) Image corrupted with Salt & Pepper noise ($prop = 0.7$)

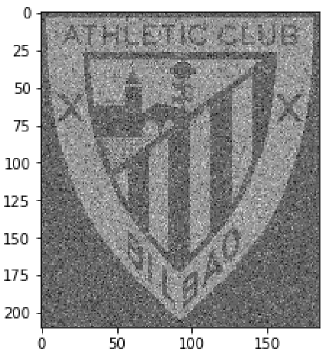

(c) Image corrupted with Gaussian noise ($prop = 0.7, \sigma^2 = 1.1$)

Figure 1: Original binary image and noisy versions

The first step was to add noise to a binary image,the results of which are shown above in Fig. 1. First, the 'salt and pepper' noise was implemented (shown in Fig. 1b), which randomly chooses a set of indices of size $prop * imgsize$, and inverts the values of those pixels in the binary picture (setting 1 to 0 and 0 to 1). Then we set the pixels to values $\{-1, 1\}$ as required by our algorithm. Following this, another version corrupted with Gaussian noise was created (shown in Fig. 1c). Similarly to the 'salt and pepper noise' implementation, we select a set of indices, however, rather than inverting the pixel values, we add a Gaussian noise sampled from $\mathcal{N}(0, \sigma^2)$.

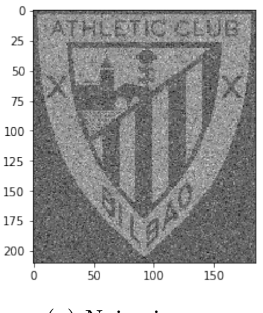
Now that the noisy images have been generated, we it is possible to apply ICM to denoise them. After observing the noisy picture ($\mathbf{y}$), and attempting to infer the value of $\mathbf{x}$, for each x_i, we check the probabilities of $p(x_i = 1|x_{\neg i}, y)$ and $p(x_i = -1|x_{\neg i}, y)$, and set x_i to 1 or -1 depending on which probability is higher. These probabilities are proportional to

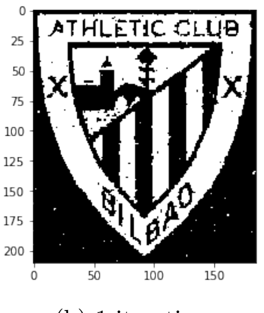
$$\prod_{i=1}^{N} e^{L_i(x_i)} e^{\sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j},$$

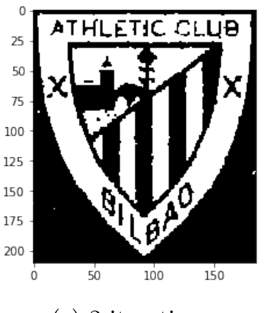
therefore we are interested in the value of $L_i(x_i) + \sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j$. In this expression, $L_i(x_i)$ characterises how the value corresponding to the noisy image contributes to the probability of x_i being 1 or -1. That is, if the noisy image is 1, it is quite likely to have come from a pixel with value 1 (particularly in the Gaussian noise version).

We can implement this by taking $L_i(x_i) = exp\{-(1 - y_i)^2\}$ for the probability of 1 and $L_i(x_i) = exp\{-(-1 - y_i)^2\}$ for the probability of -1. Thus, if y_i is close to 1 the first value will be very close to 1 and the second value quite close to 0, whereas if y_i is close to -1, it will be the other way around. The sum ($\sum_{j \in \mathcal{N}(i)} w_{ij} x_i x_j$) accounts for the relation between neighbours.

The results of the ICM de-noising are shown in Fig. 2, as can be seen, there is an improvement of the de-noising of the image from iteration 1 to 2, however following this second iteration, there is no noticeable improvement in the image.


(a) Noisy image


(b) 1 iteration


(c) 2 iterations

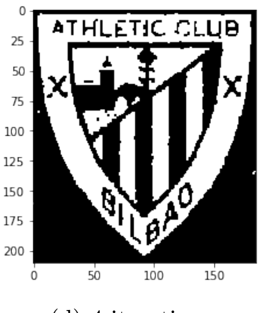
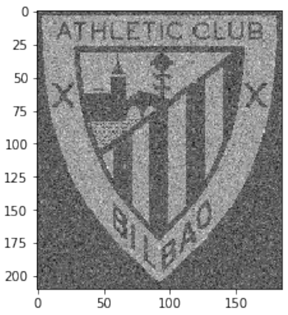

(d) 4 iterations

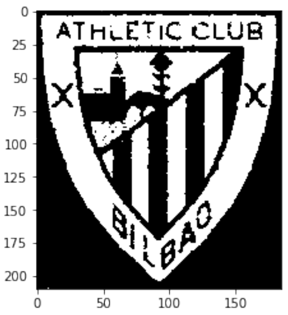
Figure 2: Results of ICM de-noising after different numbers of iterations. The image that de-noised was as shown in Fig. 1c and was generated with parameters $prop = 0.7, \sigma^2 = 1.1$.

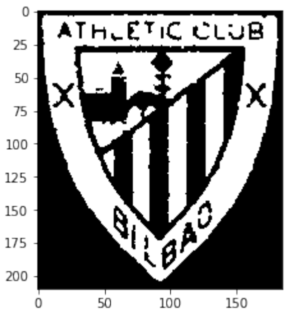
Question 2

In order to implement the Gibbs Sampler, we took $p(y_i|x_i = 1) = e^{-(1-y_i)^2}$. For the probability $p(x_i = 1, x_{\mathcal{N}(i)})$, we first sum all the neighbours (and weight this sum), then plug this sum into $e^{-(8-s)^2}$. For $p(x_i = -1, x_{\mathcal{N}(i)})$, we took $e^{-(-8-s)^2}$, where s represents the value of that weighted sum.

Some really interesting things happen when using this de-noising method. The Gibbs sampler begins de-noising promisingly, successfully de-noising the image within the first iteration. However, after the first iteration, the image is now binary instead of grey-scale, as it originally was. This happens with the Gibbs sampling as we set each pixel to a value of either 1 or -1, thus the probabilities $p(y_i|x_i)$ lose all significance, and de-noising is performed solely through the use of the values of the neighbours. This results in 'aggressive' de-noising areas of the image being overwritten and small details being lost. We can see this very clearly in Figure 3, where we used a picture with Gaussian noise with parameters $prop = 0.8, \sigma^2 = 0.8$.


(a) Noisy image


(b) 1 iteration


(c) 2 iterations

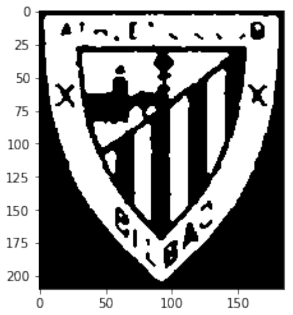
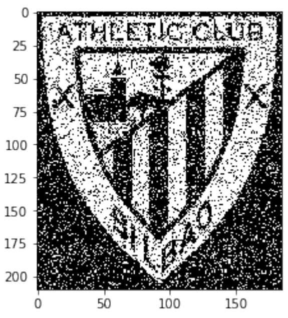
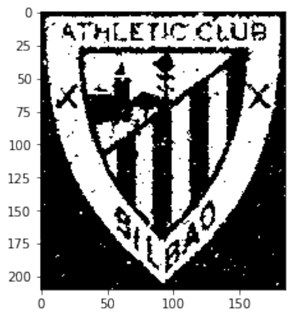

(d) 15 iterations

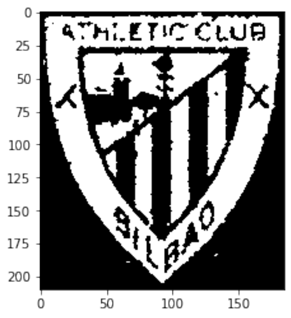
Figure 3: Results of Gibbs de-noising on an image corrupted with Gaussian noise.

In Figure 3 it is clear that after one iteration it appears to be functioning well, but it quickly becomes quite aggressive, with the neighbours taking all the significance causing deterioration of the image, particularly evident in the loss of text.

The Gibbs sampler was also implemented on an image that was corrupted with salt & pepper noise with $prop = 0.15$, as shown below in Figure 4.


(a) Noisy image


(b) 1 iteration


(c) 2 iterations

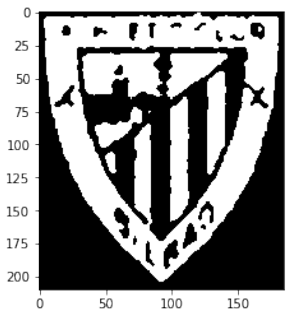
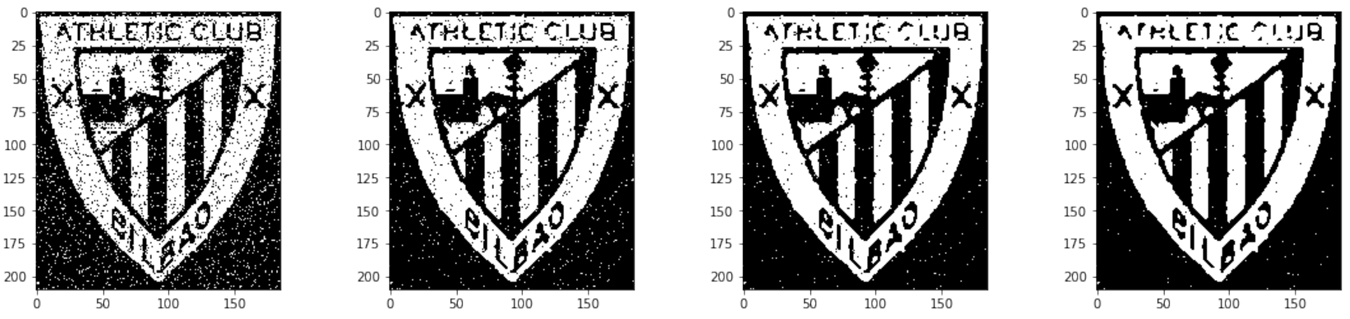

(d) 15 iterations

Figure 4: Gibbs de-noising on an image corrupted with salt & pepper noise.

From figure 4 it can be observed that the details in the image are lost after a large number of iterations. This is due to the fact that the Gibbs sampler causes the image to deteriorate.

Question 3

In order to alter the sampler so that it would pick and update a random node at each iteration, the ICM de-noising code was modified. A function was created which picks a random row and column, and then performs the ICM algorithm on that pixel of the image. We used salt & pepper noise in order to ensure that the image would always be binary. The results displayed were generated after repeating this $height \times width$ times (which represents the number of pixels in the image, which we define as one iteration), $2 \times height \times width$ times (two iterations) and $3 \times height \times width$ times (three iterations):



(a) Noisy image, S&P with $prop = 0.1$

(b) $height \times width$ nodes treated

(c) $2 \times height \times width$ nodes treated

(d) $3 \times height \times width$ nodes treated

Figure 5: ICM de-noising performed using random node locations

As we can see in Figure 5, denoising by picking random nodes results in some areas not being properly cleaned, and some others losing too much detail. Whereas in three iterations the sequential ICM cleared all the background fairly well, in 5d we see that the background is not 100% clean yet, and there are some black dots in the middle of the white areas. Additionally, the letters are more damaged than in the sequential version.

Question 4

Increasing the iterations does not always improve the result of the denoising. In the ICM method, there is a number of iterations (usually around three or four) after which no significant changes can be seen, and the picture remains somewhat stable. Nonetheless, the Gibbs method actually degenerates the pictures after a number of iterations, and small details of the picture tend to disappear as the number of iterations grow. This can be clearly seen in Figs. 3 & 4.

Taking this to the limit, when performing the Gibbs algorithm to the noisy picture displayed in Fig. 4, if we let it run for 100 iterations it almost completely removes the letters:

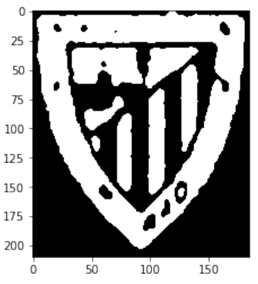


Figure 6: Gibbs sampler run for 100 iterations.

Question 5

We can consider that $KL(q(\mathbf{x})||p(\mathbf{x}))$ is an expected value of $log(Y)$, where Y is the random variable taking values $\frac{q(\mathbf{x})}{p(\mathbf{x})}$ with probability $q(\mathbf{x})$.

In a way, it is somewhat counter-intuitive that $KL(q(\mathbf{x})||p(\mathbf{x})) \neq KL(p(\mathbf{x})||q(\mathbf{x}))$, as we define KL-divergence to be the 'divergence' of q as an approximation of p. This may lead one to believe that, since q is a bad approximation of p, p is just as bad an approximation of q. This measure, however, is indeed asymmetric and in general we have

$$\begin{aligned}0 &\neq KL(q(\mathbf{x})||p(\mathbf{x})) - KL(p(\mathbf{x})||q(\mathbf{x})),\\ &= \int \Big(q(\mathbf{x})log\frac{q(\mathbf{x})}{p(\mathbf{x})} - p(\mathbf{x})log\frac{p(\mathbf{x})}{q(\mathbf{x})}\Big),\\ &= \int (q(\mathbf{x})+p(\mathbf{x}))log\frac{q(\mathbf{x})}{p(\mathbf{x})}.\end{aligned}$$

Therefore, given that expression, imagine $q(\mathbf{x})$ is a good approximation of $p(\mathbf{x})$, but is always smaller (so $q(\mathbf{x}) < p(\mathbf{x})$), and that $p(x)$ has very large values (which would be seen over a very small range as $p(x)$ must sum to 1). Then, the value that *log* evaluates is smaller than one, which yields negative numbers. Thus, and considering that $q(\mathbf{x}) + p(\mathbf{x})$ is always positive as it is the sum of a distribution function and an approximation (and actually quite a big number), we could potentially be summing a large negative amount, and the divergence would become large even if $q(x)$ was rather close from $p(x)$.

In another case, imagine that $q(x)$ is generally a good approximation of $p(x)$ but there is a region where $p(x)$ approaches zero and $q(x)$ remains slightly larger than that. Then, the value inside the *log* could grow incredibly, penalising the divergence hugely.

Question 6

We have implemented the Variational Bayes Algorithm as shown in Algorithm 4, and taking weights of $\frac{1}{8}$ or less so as to give the similar significance to the L_i's and the neighbours. An interesting thing to carry out, is to take the prior μ's as a completely random set of numbers. In our case, we tried taking μ's that are given by $\mathcal{N}(0, \sigma^2)$, and even with that, it is clear to see that the denoised image 'converges' to the original one very nicely. We take $L_i(1) = e^{-(1-y_i)^2}$ and $L_i(-1) = e^{-(-1-y_i)^2}$, and with the formulas given, find the values of m_i and μ_i, and choose according to the sign of μ. Thus, setting a noisy image with a Gaussian noise of $\sigma^2 = 0.7$ in 80% of the pixels, and setting the μ matrix to one where each item is a sample of $\mathcal{N}(0, 0.7)$, we see a nice example of how the method ends up converging and very effectively denoising the Athletic Club badge in Figure 7.

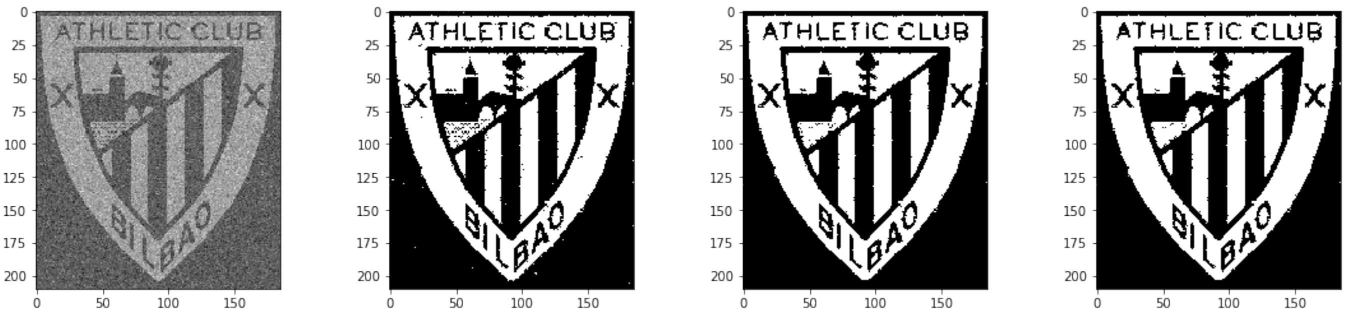

(a) Noisy image (b) 2 iterations (c) 4 iterations (d) 15 iterations

Figure 7: Results of ICM de-noising after different numbers of iterations. The image that de-noised was as shown in Fig. 1c and was generated with parameters $prop = 0.7, \sigma^2 = 1.1$.

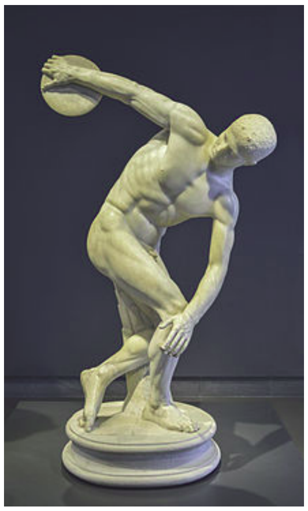
Question 7

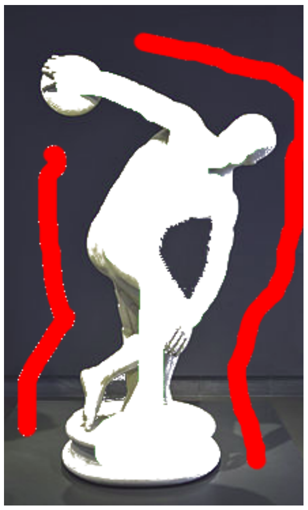
When compared to ICM and Gibbs, our Variational Bayes algorithm takes more time to achieve a total de-noising, as it has to allow the μ values converge. Even if its first iterations are a more noisy than those given by the other methods, eventually it ends up giving a very comprehensive de-noising. On the contrary, ICM and Gibbs somehow 'converged' slightly faster; however, it did not produce results this clean.

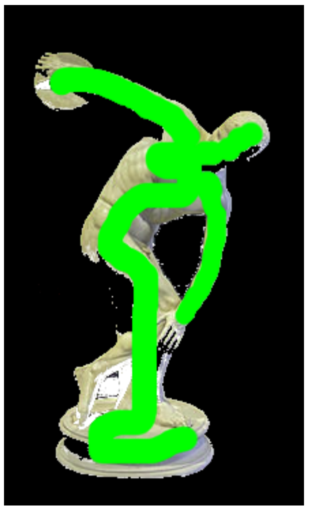
Question 8

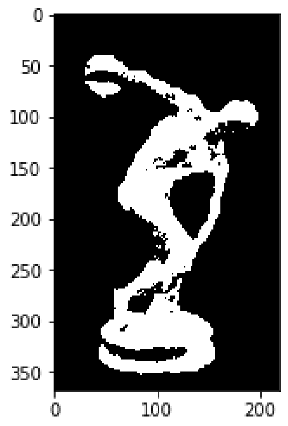
In order to implement the image segmentation, it was first necessary to distinguish the background from the foreground of the image. This was done using a mask on the image, the foreground was scribbled over with green whilst the background was scribbled over in red (as shown in Figs. 8b & 8c). Following this, histograms of the foreground and background of the image were created by manually separating the image into two images (one containing the foreground and one containing the background). The frequency of each colour (RGB) in each image was counted and stored in the two histograms which were then normalised.

These histograms were utilised in order to represent the likelihood functions which characterise how likely it is for any given pixel x_i to have come from the foreground or background, given its RBG value. This allowed to apply the Variational Bayes algorithm to classify each pixel in the original image (shown in Fig. 8a) as either having come from the foreground or background, which subsequently allowed for the image to be segmented (as shown in Fig. 8d.


(a) Original image used for image segmentation


(b) Original image with background masked and foreground removed


(c) Original image with foreground masked and background removed



(d) Segmented image generated through the use of variational Bayes

Figure 8: Image segmentation

Question 9

In the Variational Auto-encoder, parameters are updated using stochastic gradient ascent. The encoder uses a multi-layered perceptron (MLP)with a Gaussian output

In standard variational Bayes, the inference is learnt using an optimisation algorithm (Gibbs/ICM); however, computing the probability in closed forms is both slow and expensive to compute. In the VAE however, a parametric encoder is trained in order to produce parameters. A continuous random variable is generated from these parameters, and then optimised using a gradient descent algorithm.