Advanced topics in data modelling
Assignment 2: Undirected Graphical Models

Mariuta Nicolae (rqt629)

May 18, 2015

Question 1

For the implementation I have also chosen to use Matlab and I did not rely on any third party library for making the undirected graphical model. Since the algorithm is quite easy and straightforward, I also did not need any special functions from Matlab.

In the code I have introduced many comments to make it clear what I am doing at each step. The implementation begins with reading the input image, transform the pixels with value 1 into -1 and pixels with value 0 into 1. I did this because unlike the photo viewer, in Matlab the pixels appear to have opposite color (because higher pixel value means higher intensity for luminosity of pixel).

I also have defined the values of parameters *h*, *beta* and *eta* into separate variables at the beginning of the code to make it easier to change these values.

I have not used any tree data structure to represent the nodes, I only used two simple matrices X and Y to store the pairs of nodes and I just iterated through the matrix to take the pixels in order for ICM and energy calculation.

The calculation of complete energy is done in the separate function file *totalEnergy.m*; I have done like this, mostly because this function is needed in more places.

```
function  E  = totalEnergy( X,Y, h, beta, eta )
   %implementation of the complete energy function
   % based on equation E = h*sum(xi)+beta*sum(xi*xj)+eta*sum(xi*yi)

   %calculation of bias
   bias = h*sum(sum(X));

   %calculate energy for cliques formed by neighbouring nodes (xi,xj)
   cliqueNeighbours = sum(sum(X(1:size(X,1)-1,:).* X(2:size(X,1),:))) +
sum(sum(X(:,1:size(X,2)-1).* X(:,1:size(X,2)-1)));
   cliqueNeighbours = cliqueNeighbours*beta;

   %calculate energy for cliques formed by pairs (xi,yi)
   cliqueXY = eta*sum(sum(X.*Y));

   %calculate complete energy
   E = bias -  cliqueNeighbours - cliqueXY;

end
```

Figure 1. Implementation for the energy function

The function takes as input the two matrices *X* and *Y*, and also the parameters *h*, *beta* and *eta*. I just applied the function given by equation 8.3.3 in Bishop's book.directly. I calculated separately the 3 terms: the bias, the sum of all products $x_i x_j$ between the connected nodes in matrix and the sum of all pairs $x_i y_i$. At the end, the final calculation is done by doing the final sum.

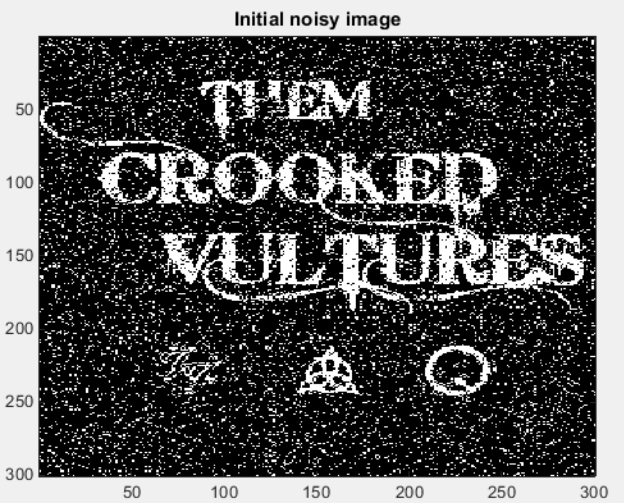
For the implementation of ICM, I have chosen to take all pixels in order by raster scan, set their value to +1 and -1, calculated the total energy for the image with both possible values of the current pixel and set the value of pixel to the value for which the total energy is the lowest.

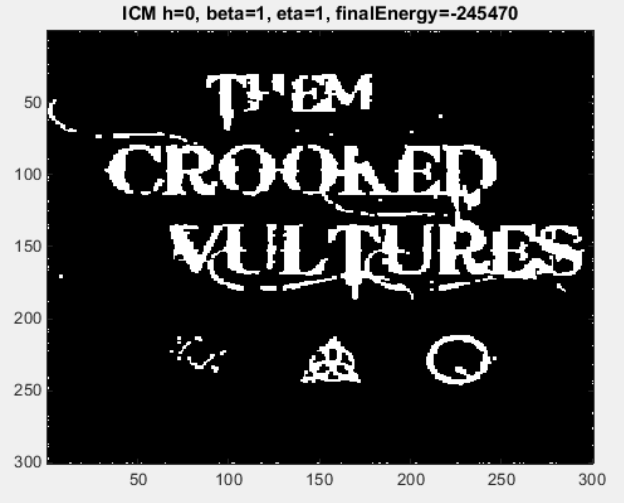
The ICM is applied several times through each pixel (I set to have 10 runs) because I have seen almost no change after first 3 runs, so it was not needed make use of the stop condition that the value of all pixels remains unchanged after a raster run.

As a conclusion, the basic algorithm is not hard to implement, but unfortunately it takes long computation time. Even if I relied as much as possible to operation with matrices instead of using for loops it still takes about 5-10 minutes to make 10 runs through all pixels. This is why for the second question; I have used the observation for the third question that the energy value of a specific pixel is only depending on the values of the surrounding pixels, and I avoided calculating the energy for all pixels at each step which saved a lot of computation time and the second implementation from my code file runs in few seconds.

Question 2

I have chosen to use the optimized implementation as described at Question 3 to be able to run the algorithm faster and I have been able to quickly see the results for changing the parameters h, beta and eta. Below I placed the original noisy image and images for various values for h, eta and beta. The plots have labels stating the values of parameters and the value of complete energy after obtaining the final result.





As it can be observed, I started by setting the values h=0, beta = 1.0 and eta = 1.0 as described in the assignment text and I got quite results. In the black space there is only little noise and on

the letters there are also some black pixels, perhaps where larger accumulations of noise appeared in the noisy image. In the case of changing value of parameter beta there is not much change in the final result which is still presenting good removal of the noise and even lower energy, mainly because, as for when the value is 1, the values of energy on vertex links $x_i x_j$ weights more than the value from the value of energy on the clique $x_i y_i$.

If eta is increased and beta is maintained at value 1, then there is more noise in the filtered image because in this situation the value of energy on the clique $x_i y_i$ weights more and, because the nodes from matrix Y contain the noisy image, with higher probability to have wrong pixel value than the surrounding pixels, because, according to the definition, the salt and pepper noise randomly switches the values of some pixels.

In the situation if increasing the bias faction h there is not much change in the resulting filtered image, but if the value is higher than 0.5, the resulting image is completely black(all pixels become -1), so it is better to keep the bias weight low in the calculation of complete energy.

Because in this situation we have unsupervised learning, it is hard to find a way to converge to the perfect solution. By only looking at the value of complete energy at the end of filtering by using this method, we cannot be sure that the result is correct because this is not and informative measure. The best way to find the optimal parameters for the ICM is to take the original photo and calculate the difference for all combinations between these parameters. But as in this situation, it is usually not possible to gain access to noise-free image to find the optimal solution. Perhaps another measure could be to count the number of edges in the filtered image and in case of good removal of salt and pepper noise, the number of edges in the final image should be lower. But this would have taken a longer time to implement.

As a conclusion, I have noticed that the results of image filtering are better if eta and beta are equal, or if beta has larger value than eta. If the value of eta is increased, making the pixels in the noisy image to weight more, then there is also more noise in the filtered image. The parameter bias is good to be kept low so it does not influence much the calculation of the complete energy.

Question 3

According to Bishop's book, the complete energy function for the model is:

$$E(x, y) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j - n \sum_i x_i y_i$$

If we only change the value of a particular node x_i , and keep all other nodes unchanged, then the difference between the values of complete energy function is the value of energy at node x_i that can be written as:

$$E(x_i, N_i, y_i) = h x_i - \beta \sum_{N_i} x_i x_j - n x_i y_i$$

The expression can be written in a better form by isolating x_i from the rest of the terms:

$$E(x_i, N_i, y_i) = x_i (h - \beta \sum_{N_i} x_j - n y_i)$$

In the equation above the parameter N_i represents the 4 neighbors of node x_i that have links to it. In the end, I obtained a sum containing two constants (h and ny_i) . And all these are multiplied by the value of current pixel x_i.

After obtaining this expression, it becomes obvious that the implementation from Question 1 can be optimized by only calculating the local energy change in $E(x_i, N_i, y_i)$, for each node in matrix X, and switch the value of pixel x_i and, again, assign to the pixel the value that obtains lower energy. I have done the optimized implementation in the second section of the code, and I used that one for answering the Question 2 because it is a lot faster and easier to observe the difference after changing the parameters values for calculating the energy.

Note that in this implementation I no longer considered the pixels on the border because it resulted into errors when taking the order of current index in the matrix if the node was on the border. Perhaps I should have done that the neighbors for the nodes on the edge of the matrix to be the pixels from the other side, because it is obvious that all pixels in the border should be black. But I have chosen to keep code simple without struggling much with such optimizations.