CS 6362 Machine Learning: Homework 5
Graphical Models
Due: 10/23/2017 11:59 PM Central Time
100 Points Total Version 1.0

1 Programming (50 points)

In this assignment you will implement a Markov Random Field for image denoising. Refer to section 8.3.3 of the book for details on how to define and update the parameters of the model. See figure 8.31 for the structure of the network. This writeup provides additional details needed for your implementation.

1.1 Image Processing Tools

As part of this assignment you will use a provided Java library for image processing, attached as `image_denoise.zip`; you should uncompress it and place the `image_denoise` directory inside your `cs362` directory. The distributed code contains three classes with main methods:

- `cs362.image_denoise.InsertImageNoise`: Insert random noise into an image by randomly permuting pixels in an image. We provide images that have been permuted, but you can use this class to create new images for testing.
- `cs362.image_denoise.RemoveImageNoise`: Remove image noise (denoise an image) using a Markov Random Field. This class calls `cs362.image_denoise.MRFImageProcessor`, which is where you will implement your MRF.

  For images with 2 colors, the two images will be compared by measuring the exact match between pixel values. If every pixel in each image contains the identical color value, the accuracy will be 100%. If no pixels in the two images have the same value, the accuracy will be 0%.

  For greyscale images, the comparison measures the average distance between the pixels in the two images. The distance between two pixels is a function of the distance between the colors of each pixel. Note that while smaller distances mean that the two images are more similar, you should not interpret the relative differences between two images compared to a baseline as success in denoising the image. In fact, an image may have a greater distance to the original image but be less noisy. You can verify your results by examining the image to see if the noise has been removed. To evaluate your code, we will compare your results to a correct implementation.

You can run each of these main methods without arguments to see a description of the command line options.

In addition to these three classes, `cs362.image_denoise.ImageUtils` contains common image processing methods that are used by multiple classes. You may find it helpful to use methods defined in this class.

You should not modify any of the above classes. All of your code should be written in `cs362.image_denoise.MRFImageProcessor`. The method `MRFImageProcessor.denoisifyImage` will decide which MRF algorithm to run depending on the input (see below). Using the descriptions below, you should check the input to see which MRF to use (i.e., check the number of colors in the image and the number of images provided.) You may check the number of colors in an image using the method `ImageUtils.countColors(int[][] image)`, which returns the number of colors in a provided image. You can also get the max color index in an image using the method `ImageUtils.maxColor(int[][] image)`.

1.2 Data

We are providing several images you may use for image processing. For each image, we provide a clean and at least one noisy version of the image. You are welcome to create more noisy versions or to use other images (make sure they are greyscale `bmp` images).

You should run `RemoveImageNoise` on the noisy image and compare the result to the image without noise. While you can use the provided class `CompareImages`, you may also find it helpful to visually inspect the images in a standard image viewer.

There are two synthetic images: easy and hard.

- **Easy**: It should be simple to remove most noise from these images and you should expect your MRF to denoisify these images almost completely correctly.
- **Hard**: These images contain random pixel values so it will be impossible for your MRF to denoisify this image. You should expect random accuracy when compared with the clean image.

While the code supports many image formats you should only use `bmp` files (extensions that end in `.bmp`). The use of other formats will cause problems with your code since various types of compression are used in other formats.

1.3 Running the Algorithm

You will implement iterated conditional modes, an iterative algorithm that updates each hidden node in the model individually. Since your output must match our implementation, you need to update the hidden nodes in a specified order. Each of your implementations should iterate over the hidden nodes as follows:

```
for (int i = 0; i < hidden_nodes.length; i++) {
    for (int j = 0; j < hidden_nodes[i].length; j++) {
        // Update hidden_nodes[i][j]
    }
}
```

where `hidden_nodes[i][j]` ($\mathbf{x}$) corresponds to the observed variable `image[i][j]` ($\mathbf{y}$). The update selects the color for hidden node i, j that minimizes the energy function. Additionally, you should initialize your hidden variables $\mathbf{x}$ to be the same as the observed variables $\mathbf{y}$.

As in the book, we will assume the bias term has the value $h = 0$ for all models in this assignment.

The run time for this code should be pretty fast. I expect about 30 seconds per iteration for the greyscale images.

1.4 Output

The output of your code should be the values of the nodes $\mathbf{x}$, which represents the denoisified image. The values for these nodes are stored in a `int[][]` object and should be returned at the end of the `MRFImageProcessor.denoisifyImage` method you will implement. You are not required to print any output in this assignment. The provided code will read and output the images.

1.5 MRFs for Black and White Images

Section 8.3.3 of the book describes a MRF for black and white images (2 pixels). When provided a single image with 2 pixels, you will use this algorithm with two small changes:

- The energy function for the clique between hidden nodes x_i and x_j is given by:

$$\psi_N(x_i, x_j) = -\beta \text{ if } x_i = x_j, \beta \text{ otherwise.}$$

- The energy function for the clique between a hidden node x_i and an observed node y_i is given by:

$$\psi_O(x_i, y_i) = -\eta \text{ if } x_i = y_i, \eta \text{ otherwise.}$$

Note that this is the same as the book except you should not expect the colors to have values $+1$ and -1. Instead, you will have different integer values as colors.

For this model, on `easy.1.2_colors.bmp` with $\eta = 2.1$, $\beta = 2.0$ and $num\_iterations = 20$ you should obtain 100% accuracy. With the same parameters on `easy.2.2_colors.bmp` you should obtain accuracy above 99.99% accuracy. The running time of both of these tests should be a few seconds.

For other images, you may experiment with different values of η and β. The values of $\eta = 2.1$ and $\beta = 1.0$ described in the book are a good starting point for exploration.

You should implement this model in `MRFImageProcessor` and it should be run when you are given a single image with only 2 colors. You may check the number of colors in an image using the method `ImageUtils.countColors(int[][] image)`, which returns the number of colors in a provided image.

This model should be used for any single image with only 2 colors.

1.6 MRFs for Greyscale Images

Extend your implementation of a single layer MRF to greyscale images. If you receive an image with more than 2 colors, assume it is a greyscale image. For greyscale images, the energy functions should be

$$\psi_N(x_i, x_j) = [\log(|x_j - x_i| + 1) - 1]\beta \ ,$$

and

$$\psi_O(x_i, y_i) = [\log(|x_i - y_i| + 1) - 1]\eta \ .$$

1.7 Additional Facts

- We will only give you valid images. Images will have arrays of uniform length (so you can iterate over them without a problem.) All colors will be non-negative integers.

- In case of a tie between two colors for maximizing the energy function, select the lowest indexed color.
- **IMPORTANT**: We will use a strategy where we fix all hidden nodes until after the full iteration is complete. This means that when you update the value of hidden node i, j, don't use this new value until the iteration is complete. This means you should save the new values and old values until you are ready to update after a full iteration.

2 Analytical (50 points)

1. (15 points) Consider the Bayesian Network given in Figure 1(a). Are the sets **A** and **B** d-separated given set **C** for each of the following definitions of **A**, **B** and **C**? Justify each answer.

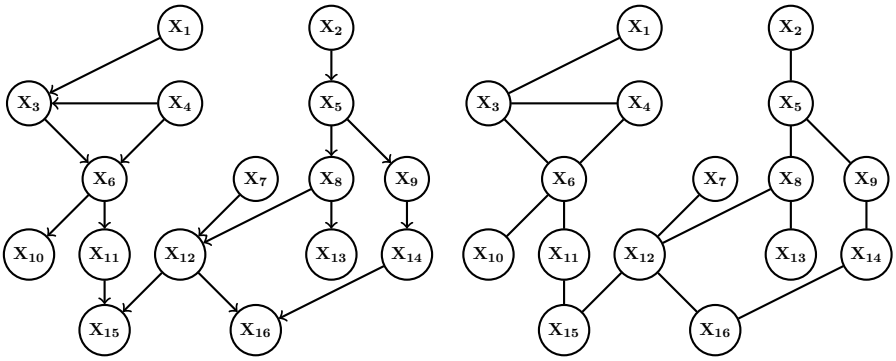
a. $\mathbf{A} = \{x_1\}$, $\mathbf{B} = \{x_9\}$, $\mathbf{C} = \{x_5, x_{14}\}$

b. $\mathbf{A} = \{x_{11}\}$, $\mathbf{B} = \{x_{13}\}$, $\mathbf{C} = \{x_1, x_{15}\}$

c. $\mathbf{A} = \{x_4\}$, $\mathbf{B} = \{x_5\}$, $\mathbf{C} = \{x_{10}, x_{16}\}$

d. $\mathbf{A} = \{x_3, x_4\}$, $\mathbf{B} = \{x_{13}, x_9\}$, $\mathbf{C} = \{x_{10}, x_{15}, x_{16}\}$

Now consider a Markov Random Field in Figure 1(b), which has the same structure as the previous Bayesian network. Re-answer each of the above questions with justifications for your answers.



(a) Directed Graphical Model (b) Undirected Graphical Model

Figure 1: Two graphs are the same. However since (a) is directed and (b) is undirected the two graphs have a different conditional independence interpretation.

2. (10 points) Let $X = (X_1, ..., X_{16})^T$ be a random vector with distribution given by the graphical model in Figure 1(a). Consider variable X_2. What is the minimal subset of the variables, $A \subset \mathcal{X} - \{X_2\}$, such that X_2 is independent of the rest of the variables $(\mathcal{X} - (A \cup \{X_2\}))$ given A? Justify your answer.

3. (25 points) The Naive Bayes model assumes that all features in a n-dimensional vector X are conditionally independent of each other given the label Y. A Naive Bayes classifier can be represented as a graphical model (figure 2). Notice that the nodes for X are shaded since they are observed. The variable Y is observed during training but is unobserved at test time.

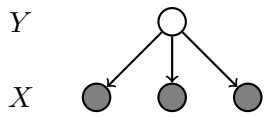


Figure 2: The graphical model for the Naive Bayes classifier.

The Naive Bayes model data likelihood is defined by:

$$L = \prod_{i=1}^{m} P(Y_i, X_i) = \prod_{i=1}^{m} \prod_{j=1}^{n} P(X_{ij}|Y_i)P(Y_i). \quad (1)$$

Now we consider a semi-supervised learning scheme with m observations, in which X is observed for all training examples, while Y is partially observed, i.e., some training examples have their Y's missing. We will call the set containing the first type of examples L (labeled) and the set containing the second type of examples U (unlabeled.) Since some of labels Y_i are missing (those in U but not those in L), we cannot maximize the joint distribution $P(Y, X)$ directly. Instead, we must maximize the likelihood function using just the examples X: $P(X)$.

Please derive an EM algorithm for maximizing the likelihood function $P(X)$ on both the labeled and unlabeled data $(L + U)$. Your solution should include the likelihood function $P(X)$ and both the E-step and the M-step of the algorithm.

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `hw5code.zip`. **You must submit source code (.java files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (hw5code.zip and hw5solutions.pdf).