Homework_6

January 3, 2020

1 Assignment 6 - STAT 7730

1.1 Problem 3

The aim of this problem is to restore a noisy image using the Ising model, which follows the formula

$$p(x) = \frac{1}{Z} \exp\left\{ \frac{1}{Temp} \sum_{<s,t>} x_s x_t \right\} \qquad x_s \in \{-1, 1\} \forall s \in S$$

where S is the 540×360 square lattice, and where $\sum_{\{<s,t>\}}$ means the sum over all pairs of neighboring sites, meaning all pairs of horizontal and vertical neighbors. Thus the neighbors of an interior pixel (r, c) are $(r+1, c), (r-1, c), (r, c+1)$, and $(r, c-1)$. Each pair only appears once in the summation. $Temp$ plays the role of temperature, so that the model strongly encourages neighboring pixels to have the same value at low temperatures, and only weakly encourages this at high temperatures.

The noisy image was obtained by taking a thresholded version of a grayscale wedding image. Note that the original version was not used but still shown in this text. At each of the temperatures \$Temp = 0.5, 1.0, \$ and 1.5 we will use the Gibbs sampler to draw an approximate sample from the "conditional" distribution of x given the corrupted image, 'y'.

[2]:
```python
import os
import numpy as np
from matplotlib import pyplot as plt
import cv2
import math
```

[3]:
```python
path = os.getcwd() + '\HW6\Wedding\\'

original = cv2.imread(path + 'Wedding', 0)
thresholded = np.loadtxt(path + 'Binary_Wedding.txt')
print("The original image (not used in this experiment) is the following:")
plt.imshow(original, cmap='gray')
plt.show()
print("\nThis image was thresholded to obtain the picture we will be working␣
↪with:")
plt.imshow(thresholded.astype(int), cmap='gray', vmin=-1, vmax=1)
plt.show()
```

```python
print("\nThis image has been corrupted by adding iid Gaussian noise")
print("N(mean = 0,sd = 1.5) to obtain the following:")

pix_1,pix_2 = thresholded.shape
Y = np.zeros((pix_1+2,pix_2+2))
Y[1:(pix_1+1),1:(pix_2+1)] = np.loadtxt(path + 'Noisy_Binary_Wedding.txt')
plt.imshow(Y, cmap='gray', vmin=Y.min(), vmax=Y.max())
plt.show()
```

The original image (not used in this experiment) is the following:

This image was thresholded to obtain the picture we will be working with:

```
This image has been corrupted by adding iid Gaussian noise
N(mean = 0,sd = 1.5) to obtain the following:
```

Let $X = (X_1, X_2, \dots, X_d)^T \sim f(x)$ be the latent binary image and $Y = ((Y_1, Y_2, \dots, Y_d)^T)$ be the observed corrupted image. Then $Y|X \sim N(X, 1.5^2 \dot{I})$.

For step 1 of the Gibbs Sampler, we can guess $X^{(0)}$ by converting into -1 all the entries with negative values and into 1 all the entries with positive values. After this, the second step would be:

$$
\begin{aligned}
f(x_s|x_{-s}, y) &\propto f(y|x)f(x) \\
&\propto N(y; x, \sigma^2) e^{\frac{1}{Temp}\sum_{t:<s,t>} x_t x_s} \\
&\propto e^{-\frac{1}{2*1.5^2}(x-y)^T(x-y)+\frac{1}{Temp}\sum_{t:<s,t>} x_t x_s} \\
&\propto e^{-\frac{1}{2*1.5^2}(x_s-y_s)^2+\frac{1}{Temp}\sum_{t:<s,t>} x_t x_s}
\end{aligned}
$$

x_s can only have two values, -1 and 1. We can see that these would be a Bernoulli random variable.

$$
f(x_s|x\_s, y) = \frac{e^{-\frac{1}{2*1.5^2}(x_s-y_s)^2 \frac{1}{Temp}\sum_{t:<s,t>} x_t x_s}}{e^{-\frac{1}{2*1.5^2}(1-y_s)^2+\frac{1}{Temp}\sum_{t:<s,t>} x_t x_s} + e^{-\frac{1}{2*1.5^2}(x_s-y_s)^2-\frac{1}{Temp}\sum_{t:<s,t>} x_t x_s}}
$$

[8]:
```
X = np.zeros((45,542,362),dtype = 'int8')

X_0 = np.zeros((542,362),dtype = 'int8')
X_0[Y < 0] = -1
X_0[Y > 0] =  1

X[0,:,:] = X_0
for temp in [0.5, 1, 1.5]:
    for i in range(1,45):
        for j in range(1,540):
            for k in range(1,360):
                if ((j+k)%2 == 1):
                    continue

                u = np.random.uniform()

                pos_x_s = float(X[i-1,j-1,k] + X[i-1,j+1,k] + \
                                X[i-1,j,k-1] + X[i-1,j,k+1])/temp;

                neg_x_s = -1*pos_x_s;
                neg_x_s = neg_x_s - ((1+Y[j,k])**2)/(2*(1.5**2));
                neg_x_s = math.exp(neg_x_s);

                pos_x_s = pos_x_s - ((1-Y[j,k])**2)/(2*(1.5**2));
                pos_x_s = math.exp(pos_x_s);

                lambd = neg_x_s/(neg_x_s + pos_x_s);

                if(lambd > u):
                    X[i,j,k] = -1;
```

```
            else:
                X[i,j,k] = 1;


        for j in range(1,540):
            for k in range(1,360):
                if ((j+k)%2 == 0):
                    continue

                u = np.random.uniform();

                pos_x_s = float(X[i,j-1,k] + X[i,j+1,k] + \
                    X[i,j,k-1] + X[i,j,k+1])/temp;

                neg_x_s = -pos_x_s;
                neg_x_s = neg_x_s - ((1+Y[j,k])**2)/(2*(1.5**2));
                neg_x_s = math.exp(neg_x_s);

                pos_x_s = pos_x_s - ((1-Y[j,k])**2)/(2*(1.5**2));
                pos_x_s = math.exp(pos_x_s);

                lambd = neg_x_s/(neg_x_s + pos_x_s);

                if(lambd > u):
                    X[i,j,k] = -1;
                else:
                    X[i,j,k] = 1;

    sample = X[44,:,:]
    print("Restored image using the Ising model with temperature %.1f" % temp)
    plt.imshow(sample, cmap='gray', vmin=-1, vmax=1)
    plt.show()
    print("")
```

```
Restored image using the Ising model with temperature 0.5
```

Restored image using the Ising model with temperature 1.0

```
Restored image using the Ising model with temperature 1.5
```

Notice that as $Temp$ is lower, it is much more likely to find nearby pixels to be equal. As $Temp$ grows, there is more flexibility for pixels to change, although this implies more noise.