Markov Random Fields

Sie Lin Wong

April 2002

Contents

1 Introduction

This tutorial discusses the use of Markov Random Fields (MRFs) in Bayesian models. If you are not familiar with probability, you might want to refer to [4], part (2) Probabilistic Models, (3) Conditional Probability and Independence, and (4) Total Probability and Bayes Rule, before reading Section 2 which explains the Bayesian models. In Section 3, some mathematical terms that will be used to explain MRFs in Section 4 are introduced. In Section 5, we discuss the implementation of MRFs and the results produced from the tests. The applications of MRFs in computer vision field are listed in Section 6. Section 7 concludes this tutorial.

The program code and results can be found in Section 8.1 (Appendix A), and some focus questions in Section 8.2(Appendix B). The references used to write this tutorial are listed in Section 9.

2 Bayesian Models

Images taken of the real world inherently contain some noise due to lighting, shadows, surface reflectance, etc. Restoring some or all features of the objects in an image using the assumptions about the world, are generally ill-posed because the data of the image may be insufficient to uniquely constrain the solution (i.e. there may exists various solutions that are compatible with the data). We called these *inverse problems*. Bayesian modeling is an approach to solve these problems.

A Bayesian model is a statistical description of an estimation problem. It consists of three components [3]:

1. *prior model*, $p(\mathbf{u})$
   a probabilistic description of the world (or its properties that we are trying to estimate) before any sensed data is collected (*a priori*).

2. *sensor model*, $p(\mathbf{d}|\mathbf{u})$
   description of the behaviour of noisy or stochastic characteristics that relate the original (unknown) state $\mathbf{u}$ to the sampled input image or sensor values $\mathbf{d}$.

3. *posterior model*, $p(\mathbf{u}|\mathbf{d})$
   probabilistic description of the current estimate of the world $\mathbf{u}$, given the observed data $\mathbf{d}$. This is calculated using Bayes' Rule:

$$p(\mathbf{u}|\mathbf{d}) = \frac{p(\mathbf{u} \cap \mathbf{d})}{p(\mathbf{d})}$$

$$= \frac{p(\mathbf{d}|\mathbf{u})p(\mathbf{u})}{p(\mathbf{d})}(\textit{since } p(\mathbf{d}|\mathbf{u}) = \frac{p(\mathbf{d} \cap \mathbf{u})}{p(\mathbf{u})}, p(\mathbf{d} \cap \mathbf{u}) = p(\mathbf{d}|\mathbf{u})p(\mathbf{u}))$$

   where $p(\mathbf{d}) = \sum_{\mathbf{u}} p(\mathbf{d}|\mathbf{u})$

To use Bayesian modeling for image processing, we need to encode a way the smoothness inherent in the image. This can be done by describing the correlation between adjacent pixels in the prior model. Markov Random Field is a way to model such correlation.

3 Neighbourhood System and Cliques

This section provides some background knowledge about neighbourhood system and cliques that would help the understanding of Markov Random Fields. For more details, you may want to refer [1].

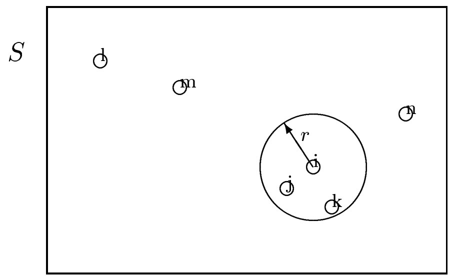

Figure 1: Neighbourhood system on a set of irregular sites

3.1 Neighbourhood System

The sites in S, in Figure 1 are related to each other through a neighbourhood system. A *neighbourhood system* for S is defined as

$$N = \{N_i | \forall i \in S\}$$

where N_i is the set of sites neighbouring site i.

The properties of neighbouring relationship are:

1. a site is not a neighbour of itself: $i \notin N_i$
2. the neighbouring relationship is mutual: $i \notin N_i \Leftrightarrow i' \in N_i$
3. the neighbouring relationship is symmetric: $i \in N_j \Leftrightarrow j \in N_i$

The neighbour set is defined as the set of nearby sites within a radius r. For example, the neighbours of site i in Figure 1 is $\{j, k\}$.

We can group neighbours into different groups by their distance from the site of interest. For regular lattice S, we call the group of the closest neighbours (sites immediately horizontally and vertically adjacent to the site of interest) the *first order neighbourhood system*, or *4-neighbourhood system*. As shown in Figure 2, in such system, an interior site has four neighbours, where x denotes the considered site and 0s its neighbours.

	0	
0	x	0
	0	

Figure 2: First order or 4-neighbourhood system

In the *second order neighbourhood system*, also called *8-neighbourhood system*, an interior site has eight neighbours (immediately horizontally and vertically adjacent, and diagonally adjacent), as shown in Figure 3.

0	0	0
0	x	0
0	0	0

Figure 3: Second order or 8-neighbourhood system

In Figure 4, the numbers $n = 1, ..., 5$ indicate the outermost neighbouring sites in the n-th order neighbourhood system.

5	4	3	4	5
4	2	1	2	4
3	1	x	1	3
4	2	1	2	4
5	4	3	4	5

Figure 4: Outermost neighbouring sites in the n-th order neighbourhood system, where $n = 1, ..., 5$

Note that the sites at or near the boundaries have fewer neighbours. For example, in first order or 4-neighbourhood system,

1. for 1D images:

   an interior site has two neighbours (Figure 5), while an end site has only one neighbour (Figure 6)

0	x	0		

Figure 5: 1D image - the neighbours of an interior site

x	0			

Figure 6: 1D image - neighbour of an end site

2. for 2D images:

   an interior site has four neighbours (Figure 7), an edge site has three neighbours (Figure 8), while a corner site has only two neighbours (Figure 9)

3.2 Cliques

A *clique* c for (S, N) is defined as a subset of sites in S. It can consist of a single site (C_1), a pair of neighbouring sites (C_2), or a triple of neighbouring sites (C_3), or so on. We can define $C_1, C_2, andC_3$ as

$$C_1 = \{i | i \in S\}$$
$$C_2 = \{\{i, i'\} | i' \in N_i, i \in S\}$$
$$C_3 = \{\{i, i', i''\} | i, i', i'' \in S \text{ are neighbours to each other}\}$$

For example, in Figure 1, for site i, $C_1 = \{i, j, k\}$, $C_2 = \{\{i, j\}, \{i, k\}, \{j, k\}\}$, $C_3 = \{\{i, j, k\}\}$

Note that the sites in a clique are ordered, i.e. $\{i, i'\}$ is not the same clique as $\{i', i\}$, etc. The collection of all cliques for (S, N) is

$$C = C_1 \cup C_2 \cup C_3 ...$$

where ... denotes possible sets of larger cliques.

For regular lattice S, in first order system, we have C_1 and C_2, as shown in Figure 10.

For second neighbourhood system, we have C_1 and C_2 as shown in Figure 10, and also C_2, C_3 and C_4 as shown in Figure 11.

	0	
0	x	0
	0	

Figure 7: 2D image - neighbours of an interior site

	0	
0	x	0

Figure 8: 2D image - neighbours of an edge site

4 Markov Random Fields (MRFs)

Random field is a set of random values $\mathbf{u} = u_1, ..., u_n$ in the set S. The random field $\mathbf{u}$ is a Markov random field on S with respect to a neighbourhood system N if and only both of the following are satisfied:

$$p(u) > 0, \forall u \in \mathbf{F} \quad \text{(positivity)}$$

$$p(u_i|\mathbf{u}) = p(u_i|\{u_j\}), j \in N_i \quad \text{(Markovianity)}$$

We can use Markovian conditional probabilities $p(u_i|\mathbf{u})$ to calculate prior model $p(\mathbf{u})$. However, it is generally hard to calculate $p(\mathbf{u})$ accuratetly. Since the distribution of Gibbs Random Fields is equivalent to the distribution of Markov Random Fields[1], we can use Gibbs distribution to calculate $p(\mathbf{u})$:

$$p(\mathbf{u}) = \frac{1}{Z} \mathrm{e}^{\frac{-E(\mathbf{u})}{T}}$$

where Z is a normalizing constant called the *partition function*

$$Z = \sum_{\mathbf{u}} \mathrm{e}^{-\frac{E(\mathbf{u})}{T}}$$

, $E(\mathbf{u})$ is the *energy function*, which can be written as a sum of *local clique energies*

$$E(\mathbf{u}) = \sum_{c \in C} E_c(\mathbf{u})$$

where each clique energy depends only on a few neighbouring sites, and T is a constant called the *temperature* of the model, which can be assumed to be 1.

Computing $p(\mathbf{u})$ using the equation above is straight forward but may be expensive due to the exponential complexity of the partition function Z. However, this computation is not necessary for most applications, so we can generate a random sample and update each u_i iteratively. Some examples of sampler algorithms that can be used to generate the random sample are *Metropolis sampling algorithm* and *Gibbs sampling algorithm*. More details about these algorithms can be found in [1].

$$p(u_i|\mathbf{u}) = \frac{1}{Z_i} \mathrm{e}^{\frac{-E(u_i|\mathbf{u})}{T}}$$

where

$$Z_i = \sum_{u_i} \mathrm{e}^{-\frac{E(u_i|\mathbf{u})}{T}}$$

0		
x	0	

Figure 9: 2D image - neighbours of a corner site

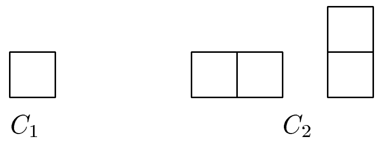

Figure 10: First order neighbourhood system

The random updating rule will converge to a representative sample from the Gibbs distribution. Simulated annealing can be used to speed up this convergence.

From $p(u_i|\mathbf{u})$ equation above, [3] shows how the following posterior distribution $p(\mathbf{u}|\mathbf{d})$ is derived using Bayes' Rule.

$$p(\mathbf{u}|\mathbf{d}) = \frac{1}{Z}\mathrm{e}^{-E(\mathbf{u})}$$

where

$$E(\mathbf{u}) = \frac{E_p(\mathbf{u})}{T_p} + E_d(\mathbf{u}, \mathbf{d})$$

To find the *Maximum A Posteriori* (MAP) estimate, i.e. the value of $\mathbf{u}$ that maximizes the conditional probability $p(\mathbf{u}|\mathbf{d})$, we only need to minimize $E(\mathbf{u})$.

5 Implementation of MRFs in a Bayesian Model

We implemented MRFs (first-order neighbourhood system and cliques) in a Bayesian model that uses Metropolis sampling algorithm (randomly initialize u_i to any value in the set S, i.e. {0,1} in this case since we are using 'binary image' only for our program) to find MAP estimate. The program reads in an image and process it using MRFs to produce an output image. It is written in Pascal — the code and results are included in Appendix A (Section 8.1).

Some constants are specified in the program to ease the coding. You can change their values in the program:

- maximum number of rows of an input image, $max\_row = 70$
- maximum number of columns of an input image, $max\_col = 70$
- total number of iteration of processing, $total\_iteration = 30$

We specify the format of the input 'image' as below:

- the first line should contain two numbers: total rows i, and total columns j ($i, j > 0$) of the input image
- the rest of the file represent the input image (of the size i rows and j columns)
- the 'images' used are 'binary images', i.e. they contain value 0 and 1 only
- $i \leq max\_row, j \leq max\_col$
- $i \geq 3$ and $j \geq 3$, so at least one pixel will have four neighbours. This is because the program is simplified — it does not process/change the edge pixel values.

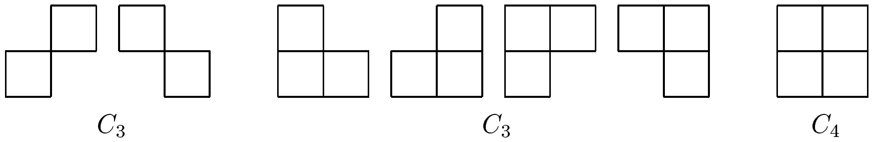

Figure 11: First order neighbourhood system

We use the following function q to decide if a randomly chosen new value should replace the current value:

$$q = \mathrm{e}^{-\frac{\Delta E}{kT}} = \frac{1}{\mathrm{e}^{\frac{\Delta E}{kT}}}$$

where
k is a constant chosen by the user at run-time
T is defined as $T = \frac{3}{\log(1+iteration\_count)}$
$E(i,j)$ is the MAP energy function for pixel (i,j)

$$E(i,j) = \frac{(4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1})^2}{T} + \frac{(O_{i,j} - u_{i,j})^2}{2}$$

with
$f_{i,j}$ = randomly chosen new value if calculating E of new value, else current value if calculating E of current value,
$f_{i-1,j}, f_{i+1,j}, f_{i,j-1}, f_{i,j+1}$ = current value of the neighbours of pixel (i,j), and
$O_{i,j}$ = original value at position (i,j)
ΔE = E of the new value - E of the current value
The new value is to replace the old value when $q \geq 1$, otherwise accept the new value with probability q when $q < 1$.

The results in Appendix A (Section 8.1) shows that the smaller k tends to remove the noise (when the area of the noise is small, e.g. a pixel whose value is different from its neighbouring pixels) and resulting in a smoother image, whereas the larger k could change the image radically if some noise is present.

The reason for this is because the larger k results in the smaller $\frac{\Delta E}{kT}$ and smaller $\mathrm{e}^{\frac{\Delta E}{kT}}$, and therefore the larger $q = \frac{1}{\mathrm{e}^{\frac{\Delta E}{kT}}}$, which means that the probability to replace the old value with a new value is greater.

This allows us to specify the degree of constraints or smoothness that we would like to have in our model.

6 Applications of MRFs in Computer Vision

Markov Random Fields provide a sound and tractable way for image analysis in Bayesian modeling. The local property (depends only on a few neighbouring sites) of MRFs also means that the algorithms can be designed in a parallel manner to speed up the calculations.

MRFs are particularly good at modeling surfaces that have certain smoothness but may not be suitable to model complicated distributions.

MRFs have been used for

- image restoration (recover 'true' scene from noisy data)
- surface reconstruction
- texture analysis
- image segmentation (break up an image into areas of similar texture)

- edge detection (locate edges corresponding to sudden changes of physical properties, e.g. surface, shape, depth, texture, etc.)
- determining discontinuities in visible surfaces
- data fusion (constructing a new image from a few images)
- solving stereo correspondence problem
- generating images of different resolution
- object matching
- object recognition
- etc.

7 Conclusion

Markov Random Fields are useful in modeling correlation between adjacent pixels in prior model of a Bayesian model. Its local property enables the estimation problems in inverse image analysis to be done in a mathematically sound and tractable way.

8 Appendix

8.1 Appendix A - Program and Results

- Program code in Pascal:

```
program mrf(input, output);

{ This program processes the original image in input file using Markov Random
  Field (MRFs), considering the neighbour above, below, on the left, and on
  the right of the pixel, and produce the output image into output file.

  It reads in input filename and output filename. Format of input file:
  (1) first line should contain two numbers: total rows (i) and total columns (j)
      (i.e. i>0 and j>0)
  (2) the rest of the file is the original image (i rows and j columns)
  (3) i<=max_row and j<=max_col
  (4) i>=3 and j>=3, so at least one pixel will have four neighbours (since the
      program is simplified - it does not process/change the edge pixel values)

  The images used in this program are binary images, i.e. contain value 0 and 1
  only.

  This program is written with Brendan McCane's help.
  Sie Lin Wong   23-Mar-2002
}

const
   max_row = 70;           { can be set to any integer }
   max_col = 70;           { can be set to any integer }
   total_iteration = 30;   { can be set to any integer }
```

```
type
   string = varying[256] of char;

var
   infile,
   outfile        : text;
   infilename,
   outfilename    : string;
   original_image,
   new_image      : array [1..max_row, 1..max_col] of integer;
   total_row,                            { total rows of original image }
   total_col      : integer;             { total columns of original image }
   k              : real;                { constant }



{ Initialize original_image and new_image, using input file. }
procedure initialize_image;
var
   i,
   j : integer;
   c : char;
begin
   open(infile, infilename, history := old, error := continue);
   reset(infile);
   read(infile, total_row);
   readln(infile, total_col);

   for i := 1 to total_row do
   begin
      for j := 1 to total_col do
      begin
         read(infile, c);
         readv(c, original_image[i, j]);
         readv(c, new_image[i, j]);
      end;
      readln(infile);
   end;

   close(infile);
end;



{ Write new_image into outfile }
procedure write_new_image;
var
   i,
   j : integer;
begin
   open(outfile, outfilename);
   rewrite(outfile);

   for i := 1 to total_row do
```

```
   begin
      for j := 1 to total_col do
         write(outfile, new_image[i, j]:1);
      writeln(outfile);
   end;

   close(outfile);
end;



{ Calculate energy }
function E(val,
           i,
           j : integer;
           T   : real) : real;
begin
   E := (sqr(4*val - new_image[i-1, j] - new_image[i+1, j] - new_image[i, j-1]
             new_image[i, j+1]))/T +
        (sqr(original_image[i, j] - val))/2;
end;



{ Process original_image using Markov random field theory to produce new_image }
procedure process_image;
var
   count,
   i,
   j,
   new_value : integer;
   T,
   delta_E,
   q         : real;
begin
   for count := 1 to total_iteration do
   begin

      { temperature }
      T := 3 / ((ln(1 + count)) / (ln(10)));   {(ln(1+count))/(ln(10))=log(1+count)}
      for i := 2 to (total_row - 1) do
      begin
         for j := 2 to (total_col - 1) do
         begin

            { randomly assign a valid value: 0 or 1 (binary image) }
            if (random < 0.5) then
               new_value := 0
            else
               new_value := 1;

            { calculate delta_E, the difference between energy of new value and
              energy of current value }
            delta_E := E(new_value, i, j, T) - E(new_image[i,j], i, j, T);
```

```
            { calculate q, acceptance probability }
            q := exp(-delta_E / (k * T));
            if (q >= 1) then
               new_image[i, j] := new_value
            else if (random <= q) then   { accept new_value with probability q}
               new_image[i, j] := new_value;
         end;
      end;
   end;
end;



BEGIN
   seed(clock);   { seeds the random number generator }
   writeln('MARKOV RANDOM FIELDS');
   write('Please enter k: ');
   readln(k);

   write('Please enter input filename: ');
   readln(infilename);
   write('Please enter output filename: ');
   readln(outfilename);

   initialize_image;
   process_image;
   write_new_image;
END.
```

- Running program:

```
> run mrf
MARKOV RANDOM FIELDS
Please enter k: 0.1
Please enter input filename: a1.in
Please enter output filename: a1.out
```

- Results:

  1. Input file:a1.in

     ```
     7 12
     000000000000
     000000000000
     000000000000
     000000000000
     010000000000
     000000001000
     000000000000
     ```

     Examples of result set:

     ```
     k = 0.1          k = 1            k = 10
     000000000000     000000000000     000000000000
     000000000000     000000000000     000111000100
     ```

```
000000000000    000000000000    001001101110
000000000000    000000000000    010011100110
000000000000    000000000000    000111010010
000000000000    010000000000    000110111110
000000000000    000000000000    000000000000
```

2. Input file: a2.in

```
7 12
100000000000
110000000000
111000000000
111100000000
111110000000
111111000000
111111100000
```

Examples of result set:

```
k = 0.1         k = 1           k = 10
100000000000    100000000000    100000000000
110000000000    110000000000    110000000000
111000000000    110001000000    101010110000
111100000000    110000000000    111000100010
111110000000    110000000000    111100101110
111111000000    111111000000    111011111000
111111100000    111111100000    111111100000
```

8.2 Appendix B - Focus Questions

1. What is a Bayesian model?
2. What is a Markov Random Field?
3. How does Markov Random Field work?
4. Why do we use Markov Random Fields in Bayesian model?
5. What are the advantages of Markov Random Fields?
6. What has Markov Random Fields been used for in computer vision field?

9 References

1. Li, S. Z. (1995). Markov random field modeling in computer vision. Springer-Verlag Tokyo.
2. Winkler, G. (1995). Image analysis, random fields and dynamic Monte Carlo methods: a mathematical introduction. Springer-Verlag Berlin Heidelberg.
3. Szeliski, R. (1999). Bayesian modeling of uncertainty in low-level vision. Digital Equipment Corporation, Cambridge Research Lab.
4. http://www.stanford.edu/class/ee178/lectures.html
5. http://cygnus.uwa.edu.au/ jphil/uni/project.html A Bayesian approach to image segmentation and image restoration / Image segmentation and reconstruction using Markov random fields
6. http://www.ucl.ac.uk/Stats/research/rr176/node25.html Markov random fields
7. McCane, B. (2002). Discussions. March-April 2002.