Computer exercise 5

Simulation and estimation for discrete Markov random process

Before you begin (on the EFD-system):
matlab, addpath(’/usr/local/mstat/matstat/init/’), fmsn20
This starts matlab and sets up the paths to the course files.

Or download the matlab files from the webpage www.maths.lth.se/matstat/kurser/fmsn20masm25/fmsn20m.zip, unpack and add the folder (with subfolders) to the matlab path. Data can be found on the webpage www.maths.lth.se/matstat/kurser/fmsn20masm25/matlab.html.

Classification of Landsat images

1. **Simple data**
   Load the data in lab6simple.dat using
   \[
   xc=\text{load}('lab6simple.dat');
   x=\text{reshape}(xc,[65,65,2]);
   \]
   This is an image with 2-dimensional pixels, each component in the range \([0,1]\), and can be visualised by \(\text{image(rgbimage(x))}\). Also look at the scatter plot of the two components, \(\text{plot}(xc(:,1),xc(:,2),'.')\) and the 2D-histogram as an intensity image, \(\text{hist2}(xc,100);\text{colormap(gray)};\).

2. **K-means**
   Use \text{kmeans.m} on the column stacked image to try to classify the pixels, first using only one component (is there any difference depending on which component you use?) and then using both components. Use the \text{plotflag} parameter to see the algorithm at work. Use \text{reshape} to view the resulting classifications.

3. **Gaussian Mixture Model**
   Fit a mixture of Normal distributions to each component of the data with the aid of \text{normmix.gibbs.m}, then fit a mixture of bivariate Normal distributions to the entire data. Use the \text{plotflag} parameter to see the algorithm at work.
   Compare the results with the K-means algorithm (with the help of \text{reshape}) both for estimates. The posterior class probabilities can be converted to a MAP estimate by \text{normmix_classify}.

4. **LANDSAT data**
   Use \text{lanread.m} to load a Landsat image.
   (rio, paris, montana, littlecoriver, or mississippi, all “.lan”, are available.)
The images are stored as \texttt{uint8} 3D-matrices in Matlab. Convert to \texttt{double} with \texttt{x=double(x)/255}. \texttt{y=x(:,:,k)} extracts image channel \texttt{k}. There are \(d = 7\) such channels in a Landsat image (see the table at the bottom of the page). Visualise the data by using the functions from Computer exercise 1, such as \texttt{rgbimage}. Note that \texttt{rgbimage} will colour the first component blue, the second green and the third red, meaning that \texttt{rgbimage(x(:,:,[1 2 3])}) will give a correctly coloured but somewhat dark RGB–image (use \texttt{landsatimage} to rescale the Landsat images).

Using the green, red and near-infrared components of the Landsat image as blue, green and red respectively will give a “false” colour (or pseudo-colour) image, as traditionally used in remote sensing to identify healthy vegetation. In the pseudo-colour image vegetation (i.e. green leaves) will be a clear red.

5. **Gaussian Mixture Model**
Fit a mixture of multivariate Normal distributions to 3 or 4 leading principal components (obtained from \texttt{pca.m}) of the data. How many different classes can you identify? Use \texttt{rgbimage} to view the posterior class probabilities.

<table>
<thead>
<tr>
<th>Landsat Channels</th>
<th>Wavelength</th>
<th>Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>450 – 520 nm</td>
<td>blue</td>
</tr>
<tr>
<td>2</td>
<td>520 – 600 nm</td>
<td>green</td>
</tr>
<tr>
<td>3</td>
<td>630 – 690 nm</td>
<td>red</td>
</tr>
<tr>
<td>4</td>
<td>760 – 900 nm</td>
<td>near–infrared</td>
</tr>
<tr>
<td>5</td>
<td>1.55 – 1.75 (\mu)m</td>
<td>infrared</td>
</tr>
<tr>
<td>6</td>
<td>10.4 – 12.5 (\mu)m</td>
<td>thermal</td>
</tr>
<tr>
<td>7</td>
<td>2.08 – 2.35 (\mu)m</td>
<td>infrared</td>
</tr>
</tbody>
</table>
Simulation of discrete Markov random fields

Let $\mathbf{x}$ be a discrete Markov random field on a regular lattice, taking values in \{1, 2, \ldots, K\}, such that the conditional distribution for $x(s_i)$ given all other points is given by

$$P(x_i = k | x_j, j \neq i) = \frac{\exp \left( \alpha_k + \beta \sum_{j \in N_i} 1(x_j = k) \right)}{\sum_{l=1}^{K} \exp \left( \alpha_l + \beta \sum_{j \in N_i} 1(x_j = l) \right)}$$

for $k = 1, 2, \ldots, K$. The neighbourhood of each point is defined by an indicator matrix $\mathbf{N}$, such as

$$\mathbf{N} = \begin{bmatrix} 0 & 1 & 0; \\ 1 & 0 & 1; \\ 0 & 1 & 0 \end{bmatrix}; \quad \text{or} \quad \mathbf{N} = \begin{bmatrix} 1 & 1 & 1; \\ 0 & 0 & 1; \\ 0 & 1 & 0 \end{bmatrix}; \quad \text{or} \quad \mathbf{N} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1; \\ 1 & 1 & 0 & 0 & 0; \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix};$$

1. **Parallel Gibbs sampling**

   Use `mrf_sim` to simulate samples from the distribution for $\tilde{\mathbf{x}}$.

   Simulate a few samples using different neighbourhood structures, with varying $\beta$ (try $-10 < \beta < 10$) and $\alpha_k = 0$, to see if the algorithm seems to work.

   Start the simulations with completely independent pixels values, for an image of size $128 \times 128$, say.

2. **Convergence**

   How many iterations does it seem to require for the simulations to converge to something with stable distribution? Display the field between each iteration, and compare the simulations for $\beta = 1$ and $\beta = 10$.

   Optional: In the simulation algorithm, compute either the full (but unnormalised) log-likelihood or the pseudo-likelihood for the simulated images, and plot as a function of the iteration count. How long does it take for the curve to “stabilize”?

   Calling `mrf_sim` with zero iterations computes the number of neighbours $M_f$, eg:

   $[z, Mz, Mf] = \text{mrf_sim}(z0, N, alpha, beta, 0);$