Mixture Models and Their Applications

text for students on MMJ

Contents

1 Estimation of single models 2
  1.1 Normal regression model ........................................... 2
  1.2 Categorical model with dynamics .............................. 3

2 Mixture model and its estimation 4
  2.1 Model .......................................................... 4
  2.2 Estimation .................................................... 7

3 Classification with the mixture model 11
  3.1 Standard way of classification .................................. 11
  3.2 Other variants of classification ............................... 11

4 Prior setting of mixture estimation 14
  4.1 Scale data and set random centres .............................. 14
  4.2 Use prior data and pre-estimate ............................... 14
  4.3 Use expert knowledge and pre-estimation ..................... 15
  4.4 Estimation of component variances ............................ 15

5 Validation of the estimation 15

6 Appendix 16
  6.1 Bayes formula .................................................. 16
  6.2 Chain rule ..................................................... 16
  6.3 Derivation of mixture estimation algorithm .................... 16
  6.4 Least square estimation ....................................... 18
  6.5 Estimation of model order ..................................... 19
  6.6 Initialization of mixture estimation ............................ 20
1 Estimation of single models

The model in Bayesian statistics has a form of conditional probability density function (pdf) of the following form

\[ f(y_t | \psi_t, \Theta) \]  

(1.1)

where \( y_t \) is the system output, \( \psi_t \) is the regression vector containing samples on variable on which the output depends and \( \Theta \) is a collection of model parameters. This pdf can be constructed in a variety of ways, according to the model under consideration.

1.1 Normal regression model

For the normal regression model the pdf (1.1) is constructed by the regression equation

\( y_t = \psi_t \theta + e_t \)

where \( e_t \) is a white noise with zero expectation and constant variance \( r \). E.g. for the first order model it is

\( \psi_t \theta = [y_{t-1}, u_t, u_{t-1}, 1] [a_1, b_0, b_1, k]^\prime \).

Estimation

\( \Psi_t = [y_t, \psi_t]^\prime \)

\( V_t = V_{t-1} + \Psi_t \Psi_t^\prime \)

\( \kappa_t = \kappa_{t-1} + 1 \)

with prior statistics \( V_0 \) and \( \kappa_0 \).

Points estimates

\( V_t = \begin{bmatrix} V_y & V_{y\psi}^\prime \\ V_{y\psi} & V_{\psi} \end{bmatrix} \)

\( \hat{\theta}_t = V_{\psi}^{-1} V_{y\psi} \)

\( \hat{r}_t = \frac{V_y - \hat{\theta}_t V_{y\psi}}{\kappa_t} \)

Example - dynamic model

Model

\( y_t = a_1 y_{t-1} + b_0 u_t + b_1 u_{t-1} + k + e_t \)

Extended regression vector

\( \Psi_t = [y_{t-1}, u_t, u_{t-1}, 1]^\prime \)

Information matrix

\( V_t \sim 4x4, \quad V_y \sim 1x1, \quad V_{y\psi} \sim 3x1, \quad V_{\psi} \sim 3x3 \)

\( (~ \text{denotes dimensions of a matrix}) \)

Update and point estimates are as indicated above.
Example - static model

Model

\[ y_t = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{bmatrix} + e_t \]

Extended regression vector

\[ \Psi = [y_1, y_2, y_3, y_4, 1] \]

Information matrix

\[ V_t \sim 5 \times 5, \quad V_y = 4 \times 4, \quad V_{y\psi} \sim 4 \times 1, \quad V_\psi = 1 \times 1 \]

Batch estimation using least squares

A very efficient way of estimating a regression model from an already measured data sample \( \{y_t, u_t\}_{t=1}^T \) is to use the least squares (LS) method. The LS method is described in Appendix 6.4 on page 18.

Estimation of the model order

To prior estimation also determination of the model used for estimation belongs. It can be done very easily.

1. Normalize the data used for estimation so that their average is zero and variance is one.
   I.e. for a variable \( x \) compute \( \xi \) so that
   \[ \xi = \frac{x - \bar{x}}{\sqrt{r_x}} \]
   where \( \bar{x} \) is average and \( r_x \) is variance.
2. Perform regression analysis with some maximal model order (say 10).
3. Plot the regression coefficients in a bar graph. Here you can divide the coefficients to very small (almost zero) and the rest. The variables corresponding to “zero” coefficients can be omitted from the regression vector.

A sample program can be found in Appendix 6.5 on page 19.

1.2 Categorical model with dynamics

The pdf (1.1) is parametrized directly \( f(y_t|\psi_t, \Theta) = \Theta_{y_t|\psi_t} \) where \( y_{t|\psi_t} = [\psi'_t, y_t] \) is a multi-index and the sign \( | \) is only formal (the multi-index denotes a transposed matrix).

The model can be expressed in a form of table (for \( \psi_t = [y_{t-1}, u_t] \) and binary variables)
<table>
<thead>
<tr>
<th>$y_{t-1},u_t$</th>
<th>$y_t = 1$</th>
<th>$y_t = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>$\Theta_{1</td>
<td>11}$</td>
</tr>
<tr>
<td>1,2</td>
<td>$\Theta_{1</td>
<td>12}$</td>
</tr>
<tr>
<td>2,1</td>
<td>$\Theta_{1</td>
<td>21}$</td>
</tr>
<tr>
<td>2,2</td>
<td>$\Theta_{1</td>
<td>22}$</td>
</tr>
</tbody>
</table>

**Estimation**

The statistics is $\nu_{i|jk}$ and has the same form as the model.

$$\nu_{y_t|\psi;t} = \nu_{y_t|\psi;t-1} + 1$$

i.e. the entry, corresponding to the actual data item $y_t, \psi_t$ is increased by one.

The prior statistics is $\nu_0$.

**Point estimates**

are given by the normalized statistics (each row is divided by its sum)

$$\hat{\Theta}_{i|jk;t} = \frac{\nu_{i|jk;t}}{\sum_l \nu_{l|jk;t}}$$

**Example**

For the statistics

<table>
<thead>
<tr>
<th>$y_{t-1},u_t$</th>
<th>$y_t = 1$</th>
<th>$y_t = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>15</td>
<td>8</td>
</tr>
<tr>
<td>1,2</td>
<td>21</td>
<td>35</td>
</tr>
<tr>
<td>2,1</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2,2</td>
<td>31</td>
<td>5</td>
</tr>
</tbody>
</table>

the point estimates of parameters are

<table>
<thead>
<tr>
<th>$y_{t-1},u_t$</th>
<th>$y_t = 1$</th>
<th>$y_t = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,1</td>
<td>0.652</td>
<td>0.348</td>
</tr>
<tr>
<td>1,2</td>
<td>0.375</td>
<td>0.625</td>
</tr>
<tr>
<td>2,1</td>
<td>0.312</td>
<td>0.688</td>
</tr>
<tr>
<td>2,2</td>
<td>0.861</td>
<td>0.139</td>
</tr>
</tbody>
</table>

where the first row is divided by 23, second one by 56, then 32 and finally 36.

**2 Mixture model and its estimation**

2.1 Model

Model of the mixture is used for a description of multi-modal systems that switch between a finite amount of different working modes. It is composed of (i) a set of ordinary models (here normal regression or categorical) and (ii) a pointer model - its output is a discrete random process (a sequence of discrete random variables) whose values at each time instant point to the active component.
Example

Let us have a mixture with two static components and static pointer model (i.e. neither components nor the pointer model depend on past values of their outputs). The components have two dimensional outputs.

Component 1 - \( f_1(y_t|k) \)

\[
\begin{bmatrix}
  y_{1,t} \\
  y_{2,t}
\end{bmatrix} = \begin{bmatrix} 8 \\ 1 \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix}
\]

Component 2 - \( f_2(y_t|k) \)

\[
\begin{bmatrix}
  y_{1,t} \\
  y_{2,t}
\end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \end{bmatrix}
\]

Model of the pointer \( f(c_t|\alpha) = \alpha_{c_t} \)

\[
\begin{array}{c|cc}
  c_t & 1 & 2 \\
  \alpha_{c_t} & 0.2 & 0.8 \\
\end{array}
\]

The simulation with this model is performed by the following program and the result of the simulation is in the picture.

```matlab
// Estimation of a simple mixture
clc, clear, close, close
[u,t,n]=file();
chdir(dirname(n(1)));
getd();

load simple.dat ct yt
nd=max(size(yt));
mu=rand(2,1,'u');
al=mu/sum(mu);

V=list(); thE=list(); cvE=list();

thE(1)=[6;2]; thE(2)=[1;3];

for j=1:2
    V(j)=[thE(j);1]*[thE(j);1]';
    cvE(j)=.1*eye(2,2);
end

ka=rand(1,2,'u');

for t=1:nd
    // computation of weights
    for j=1:2
        [xxx,mL(j)]=GaussN(yt(:,t),thE(j),cvE(j));
    end
    mp=sum(mL);
    mL=mp./max(mL);
end
```

5
\( m = \exp(mp) \); \hspace{1cm} // exponent
\( w = m \cdot a1; \hspace{1cm} w = w/\text{sum}(w); \hspace{1cm} // \text{component weights}
\( \text{wt}(:,t) = w; \hspace{1cm} // \text{store} \)

\text{recomputation of statistics} 
\( \text{Ps} = [y(:,t);1]; \)
\text{for} \ j = 1:2
\( \text{V}(j) = V(j) + w(j) \cdot \text{Ps} \cdot \text{Ps}'; \)
\( \text{ka}(j) = \text{ka}(j) + w(j); \)
\text{end}

\text{point estimates} 
\( \text{thE}(j) = (\text{inv}(V(j)(3,3)) \cdot V(j)(3,1:2))'; \)
\text{if} \ t > 50
\( \text{cvE}(j) = (V(j)(1:2,1:2) - \text{thE}(j) \cdot V(j)(3,1:2)) / \text{ka}(j); \)
\text{end}
\text{end}
\( \text{nu} = \text{nu} + w; \hspace{1cm} // \text{pointer-model statistics} \)
\( \text{al} = \text{nu}/\text{sum(nu)}; \hspace{1cm} // \text{and parameter} \)

\( \text{th1}(:,t) = \text{thE}(1); \hspace{1cm} // \text{store for} \)
\( \text{th2}(:,t) = \text{thE}(2); \hspace{1cm} // \text{plot} \)
\text{end}

\text{Results} 
\( \text{scf(1);} \)
\( \text{plot(wt')} \)

\text{set(scf(2),'position',[900 100 800 600])} 
\text{subplot(211),plot(th1')} 
\text{subplot(212),plot(th2')} 
\text{save estim.dat}
The picture shows a scatter-plot of two dimensional mixture with two components. The output forms two clusters. Each one represents one working mode of the system.

In the program we used the subroutine \texttt{GaussN} that computes a value of multinomial normal distribution. This subroutine is here

\begin{verbatim}
function \[p,Lp\]=GaussN(x,m,R)
    // \[p \ \text{Lp}\]=GaussN(x,m,R) value of multivariate Gaussian pdf
    // p probability
    // Lp logarithm of prob.
    // x realization
    // m expectation
    // R covariance matrix
    x=x(:); // column vector
    m=m(:); // column vector
    n=max(size(R));
    Lp=-.5*(n*log(2*%pi)+log(det(R)));
    ex=(x-m)'*inv(R+1e-8*eye(n,n))*(x-m);
    Lp=Lp-.5*ex;
    p=exp(Lp);
endfunction
\end{verbatim}

2.2 Estimation

We estimate

1. parameters of all components \((\theta, r, \text{resp. } \beta)\),
2. parameters of pointer model \((\alpha)\),
3. current values of the pointer.

The mixture estimation without the estimation of the active component would not work. It is necessary that each individual component is assigned to one working model of the system. It means that before the mixture estimation (update of statistics of components and the mixture model) it is necessary to decide to which component the measured data item belongs and to update the statistics just for this component. In practice, we do not know, which working model generated the data item. So we have to estimate it. The uncertainty causes, that there is no one hundred percent active component but we assign probabilities of activity to each component and with those probabilities (in the meaning of a portion from the whole data measured) we use the data item for the update.

The resulting algorithm of the mixture estimation is following

1. Measure new data item.
2. Estimate the current pointer value
   (a) determine “proximity” of the data item to individual components
   (b) compute the weighting vector \(w\) as a product of the proximity and component probability (expressed as relative frequency of past activities of individual components).
3. Recompute statistics of all components and pointer model with the measured data item multiplied by the corresponding item of the vector \(w\).
4. Determine the point estimates of all unknown parameters.

Comment

1. It is clear.
2. To compute the proximity, we substitute into the component model the existing point estimates of its parameters (i.e. those from the last step of estimation) as well as the newly measured data item. This value, called proximity, is the bigger the closer the data are to the component centre.

   The component probabilities are constructed as relative frequencies from the past estimates of component activities.

   The product of proximity and the relative frequency of the component activity gives the corresponding item of the vector which after normalization to sum equal to one is equal to the weighting vector \(w\).

   These computation depend on point estimates of parameters from the last step of estimation.

3. The update of statistics is the same as that for the individual models (the continuous model updated the information matrix and the counter; the discrete model increments the statistics item corresponding to the measured data). The only difference is that the data item entering the update is multiplied by the corresponding item of the weighting vector \(w\).
4. The point estimates of parameters are performed in exactly the same manner as for the individual models. They are prepared for the use in the following step of estimation.

In our example we have:

Components are Gaussian distributions with regression vectors

\[ \Psi_t = [y_{1:t}, y_{2:t}, 1] \]

and each component has its own statistics

\[ V_{i;t} = \begin{bmatrix} V_{11} & V_{12} & V_{13} \\ V_{21} & V_{22} & V_{23} \\ V_{31} & V_{32} & V_{33} \end{bmatrix} = \begin{bmatrix} V_y [2x2] & V_{y\psi} [2x1] \\ V_{y\psi} [2x2] & V_y \end{bmatrix} \]

\[ \kappa_{i;t} = [\kappa_1, \kappa_2] \]

\( i = 1, 2 \).

Update of statistics is

\[ V_{i;t} = V_{i;t-1} + w_i \Psi_t \Psi_t' \]

\[ \kappa_{i;t} = \kappa_{i;t-1} + w_i \]

\( i = 1, 2 \).

Point estimate

\[ \hat{\theta}_i = sV^{-1}_{\psi}V'_{y\psi} \quad \hat{r}_i = \frac{V_y - \hat{\theta}V_{y\psi}}{\kappa_i} \]

\( i = 1, 2 \).

The pointer \( c_t \) is a discrete stochastic process with two possible values \( c_t \in \{1, 2\} \).

Model of the pointer is a categorical static distribution with a binary output

\[
\begin{array}{c|cc}
   c_t & 1 & 2 \\
   \hline
   f(c_t | \alpha) & \alpha_1 & \alpha_2 \\
\end{array}
\]

Its statistics has the same form as the model

\[
\begin{array}{c|cc}
   c_t & 1 & 2 \\
   \hline
   \nu_t & \nu_1 & \nu_2 \\
\end{array}
\]

Statistics update is

\[ \nu_{i;t} = \nu_{i;t-1} + w_i \]

\( i = 1, 2 \).

Point estimate

\[ \hat{\alpha}_t = \frac{[\nu_{1;t}, \nu_{2;t}]}{\sum \nu_{i;t}} \]

Remark

In practical computations, the proximities (values of the estimated component distributions in measured data) must be computed in logarithm, then normalized by subtracting the maximal
value and only then taking the exponential of them. They can (and usually are) very small numbers and if computed directly they are frequently out of the range of computer word.

The estimation is illustrated in the following program

```matlab
// Estimation of a simple mixture
clc, clear, close, close  // clear all
[u,t,n]=file();  // find working directory
chdir(dirname(n(1)));  // set working directory
getd();

load simple.dat ct yt // load of data (from SimpleMixSim)
nd=max(size(yt));  // length of data
mu=rand(2,1,'u');  // pointer-model statistics
al=mu/sum(mu);  // pointer-model parameter

V=list(); thE=list(); cvE=list();
thE(1)=[6;2];  // in simulation was 8.1 initial
thE(2)=[1;3];  // in simulation was 0.5 parameters
for j=1:2
    V(j)=[thE(j);1]*[thE(j);1]';  // initial inf. matrix
    cvE(j)=.1*eye(2,2);  // initial noise covariance
end

ka=rand(1,2,'u');  // data counter

for t=1:nd // time loop
    // computation of weights
    for j=1:2
        [xxx,mL(j)]=GaussN(yt(:,t),thE(j),cvE(j));  // proximity
    end
    mp=mL-max(mL);  // normalization
    m=exp(mp);  // exponent
    w=m.*al; w=w/sum(w);  // component weights
    wt(:,t)=w;  // store

    // recomputation of statistics
    Ps=[yt(:,t);1];
    for j=1:2
        V(j)=V(j)+w(j)*Ps*Ps';
        ka(j)=ka(j)+w(j);
    end

    // point estimates
    thE(j)=(inv(V(j)(3,3))*V(j)(3,1:2))';
    if t>50
        // in the beginning, cvE is fixed
        cvE(j)=(V(j)(1:2,1:2)-thE(j)*V(j)(3,1:2))/ka(j);
    end
end
```
nu = nu + w;             // pointer - model statistics
al = nu / sum(nu);      // and parameter

th1(:,t) = thE(1);      // stor for
th2(:,t) = thE(2);      // plot
end

// Results
scf(1);
plot(wt')

set(scf(2), 'position', [900 100 800 600])
subplot(211), plot(th1')
subplot(212), plot(th2')
save estim.dat

3 Classification with the mixture model

The mixture estimation is performed in two steps: (i) classification and (ii) estimation. The classification is a part of estimation.

3.1 Standard way of classification

1. Learning - the estimation with a data sample and known active component to each data item. Here the estimation is simple. Knowing the active components, each data item can be assigned deterministically to its active component and only this component is estimated at this time instant. I.e. the components are estimated independently.

2. Testing - only the classification is performed, i.e. just the weighting vector \( w \) is constructed. The components are not estimated and stay in the same form as they were after the learning phase.

3.2 Other variants of classification

As the classification is a part of the broader task of mixture estimation there is a lot of other variants for the classification algorithm

1. If at the beginning of classification we do not have a learning data sample, we can start mixture estimation with only a prior information. During the estimation we do the classification as well, however, at the beginning the estimation only starts, so the model is not precise and it only learns as the data are coming. So, at the beginning also the classification can work with less precision.
2. If we are in the testing phase (we have learned the model and now we deal with data without knowledge of the active components) and from time to time we learn the active component, we can use this knowledge and to update the the active component directly (with vector \( w \) full of zeros and with one on the position of the active component).

3. On the other hand, we can use these reasonings for prior estimation of mixtures. This is discussed in Chapter 4

The mixture classification is demonstrated in the following programs:

1. Learning

```matlab
// Learning a simple mixture for classification
clc, clear, close, close             // clear all
[u,t,n]=file();                      // find working directory
chdir(dirname(n(1)));               // set working directory
getd();

load simple.dat ct yt                // load of data (from SimpleMixSim)
nd=max(size(yt));                    // length of data

mu=.001*rand(2,1,'u');              // pointer-model statistics
V=list(); thE=list(); cvE=list();    // initial inf. matrix
for j=1:2
    V(j)=.001*eye(3,3);              // initial inf. matrix
end
ka=.001*rand(1,2,'u');              // data counter

for t=1:nd                          // time loop
    wt=zeros(2,1); w(ct(t))=1;     // known active component

    // recomputation of statistics
    Ps=[yt(:,t);1];
    for j=1:2
        V(j)=V(j)+w(j)*Ps*Ps';
        ka(j)=ka(j)+w(j);

        // point estimates (only for plot)
        thE(j)=(inv(V(j)(3,3))*V(j)(3,1:2))';
    end
    nu=nu+w;                         // pointer-model statistics

    th1(:,t)=thE(1);                 // for
    th2(:,t)=thE(2);                 // plot
end
al=nu/sum(nu);                      // and parameter
for j=1:2
    cvE(j)=(V(j)(1:2,1:2)-thE(j)*V(j)(3,1:2))/ka(j);
```
end

// Results
set(scf(1), 'position', [900 100 800 600])
subplot(211), plot(th1')
subplot(212), plot(th2')
save learn.dat

2. Testing

// Testing of a simple mixture
// connected with simpleMixSim.sce and simpleMixLearn.sce
// data for testing are also from simpleMixSim
clc, clear, close, close // clear all
[u, t, n] = file(); // find working directory
chdir(dirname(n(1))); // set working directory
getd(); mode(0)
load learn.dat // load results of learning
load simple2.dat yt ct // load data for testing

for t = 1:nd // time loop
    // computation of weights
    for j = 1:2
        [xxx, mL(j)] = GaussN(yt(:, t), thE(j), cvE(j)); // proximity
    end
    mp = mL - max(mL); // normalization
    m = exp(mp); // exponent
    w = mL * a1; w = w/sum(w); // component weights
    wt(:, t) = w; // store
end

// Results
[xxx, ce] = max(wt, 'r'); ce = ce'; // point estimates of pointer
wrong = sum(ct ~= ce) // number of wrong classifications
from = nd

s = 1:100;
set(scf(1), 'position', [900 100 800 600])
plot(s, ct(s), 'or', 'markersize', 14)
plot(s, ce(s), 'b', 'markersize', 10)
set(gca(), 'data_bounds', [1 max(s) min(ct) - .2 max(ct) + .2])
title 'Testing of simple mixture estimation'
4 Prior setting of mixture estimation

No mixture can be estimated without its proper initialization. The importance of initial setting of estimation lies in the fact that individual components of the mixture model must be assigned to individual system modes. If all prior components are the same, their separation is not possible (all of them follow the current data). On the other side, if some prior component is placed too far from the area where the data lay, its weights will be so small that it remains in its prior position and is excluded from estimation. So, good initialization is a necessary basis for any mixture estimation.

4.1 Scale data and set random centres

For a variable $x$ we determine the scaled variable $\xi$ which has zero average and unit variance in the following way

$$\xi = \frac{x - \bar{x}}{\sqrt{r_x}}$$

where $\bar{x}$ is average of $x$ and $r_x$ its variance. After scaling, the dominant part of data lies within the range $\pm 3$. Thus the area of data is given and here should be located the initial centres of the prior components.

After the analysis, the results are valid for the scaled data. If we want to return back to the original scales, we can use the inverse formula

$$x = \sqrt{r_x} \xi + \bar{x}.$$
4.3 Use expert knowledge and pre-estimation

If the data are not at disposal, we can use an expert knowledge to form some fictitious data, i.e. such data which will be generated by a system possessing a property, that corresponds to the expert information.

Example

We investigate traffic accidents (density) in dependence on the speed of the vehicle [km/h] and daily time [time]. We have no prior data at disposal, so we ask an expert. He insist, that mostly accidents happen in the morning about 8 o’clock when cars going too fast, say 80 km/h and then afternoon approximately between 17 and 19 o’clock even in much lower speed, about 50 km/h, when drivers are tired after their working hours.

The information obtained from the expert says, that data typical for our system are couples [80, 7] (generated by the first component) and [50, 18] (belonging to the second one).

Appendix 6.6 shows, how to deal those fictitious data.

4.4 Estimation of component variances

Estimating of variances more precisely covariance matrices of components is much more difficult then the expectations (centres). However, their precise estimates are not so important, because mostly we are interested in the component positions.

Moreover, the “full” estimation of the component variances is dangerous. At the beginning of estimation when the component positions are not correct, there is a danger that the variance of one component grows rapidly and this component covers all data clusters. The remaining components stay unestimated and the result of estimation is a failure. Thus, we can either

(i) fix the covariances and leave them their initial values (e.g. for normalized data with variances one the covariances can be set as 0.1*eye(dim,dim), where dim stays for the dimension of data or

(ii) to fix the covariances at the beginning of estimation (say 30-50 steps) and to start their estimation after this period when the covariances are better located.

5 Validation of the estimation

1. The first indicator of good estimation can be viewed from the values of the vector statistics \( \kappa \). Its values reflect frequencies of activities of individual components. If some entries are zero or close to zero, the respective components were not estimated and remained “empty”.

2. Similar indicator is given by the evolution of the pointer estimates in the course of estimation. The components should be switched in a reasonable manner (according to the specific example). Also, no component should be excluded from switching and stay constant.

3. More precise confirmation of good estimation is an evolution of the point estimates of centres of all components. The resulting values also indicate the final positions of the components.
4. The most important validation of successfulness of mixture estimation is the analysis of prediction error, which is difference of measured output and that predicted (estimated) from the model. For two-dimensional output the real data should form approximately the same clusters as predictions. The sum of prediction errors should be minimal. The prediction errors should have zero average and the variance corresponding to the expected noise variance.

6 Appendix

6.1 Bayes formula

Generally, the estimation obeys the Bayes rule in this form

\[ f(\Theta | d(t)) \propto f(d_t | \psi_t, \Theta) f(\Theta | d(t-1)) \]

6.2 Chain rule

The joint pdf can be decomposed as follows (e.g. for data)

\[ f(d(t)) = f(d_t | d(t-1)) f(d_{t-1} | d(t-2)) \cdots f(d_1 | d_0) f(d_0) \]

6.3 Derivation of mixture estimation algorithm

The theoretical derivation of estimation with mixture model starts with the posterior pdf - i.e. the pdf of all unknown objects, which are parameters of all components \( \Theta = \{ \Theta_i \}_{i=1}^n \), parameters of the pointer model \( \alpha \) and the pointer itself \( c_t \) which is conditioned on data for which the Bayes rule can be used

\[ f(c_t, \alpha, \Theta | d(t)) \propto f(d_t, c_t, \alpha, \Theta | d(t-1)) \]

Bayes

The pdf on the right-hand side can be decomposed using chain rule

\[
\begin{align*}
    f(d_t, c_t, \alpha, \Theta | d(t-1)) &= f(d_t | c_t, \alpha, \Theta, d(t-1)) f(c_t | \alpha, \Theta, d(t-1)) f(\alpha | d(t-1)) f(\Theta | d(t-1)) \\
    &= \underbrace{f(d_t | c_t, \psi_t, \Theta)}_{(i)} \underbrace{f(c_t | \alpha)}_{(ii)} \underbrace{f(\alpha | d(t-1))}_{(iii)} \underbrace{f(\Theta | d(t-1))}_{(iv)}
\end{align*}
\]

where the missing variables in the above conditions are assumed independent of the variables before the condition sign \(|\).

Now, the pdfs on the right-hand side are (i) model of the \( c_t \)-th component, (ii) model of the pointer, (iii) prior for estimation of the component, (iv) prior for estimation of the pointer model.
To be able to update the above formula, i.e. to obtain updated formula $f(\alpha, \Theta | d(t))$ from the formulas $f(\alpha | d(t-1))$ and $f(\Theta | d(t-1))$ which follow from the previous step of estimation, we would need to marginalize the pdf $f(c_t, \alpha, \Theta | d(t))$, i.e. to compute

$$f(\alpha, \Theta | d(t)) = \sum_{c_t} f(c_t, \alpha, \Theta | d(t))$$

which produces the posterior pdf in the form of a sum. In the Bayes rule the posterior is repetitively multiplied by the model pdf. In this case we would produce product of sums, which is computationally unfeasible (the expression obtained are so complex that they cannot be evaluated in a reasonable time). That is why we need to approximate.

We start with the assumption that the active component is known. Then, the probability function of the pointer is a Kronecker function $\delta(c_t, \vec{c})$ (vector of zeros with one at the specified point $\vec{c}$)

$$f(c_t | d(t)) = \delta(c_t, \vec{c}) \quad \text{(6.2)}$$

where $\vec{c}$ is the known value of the actually active component. However, the assumption of our knowledge of the active component is not valid ($\vec{c}$ is not known to us). So instead of the values of $\delta(c_t, \vec{c})$ we take its expectation

$$\delta(c_t, \vec{c}) \rightarrow E[\delta(c_t, \vec{c}) | d(t)] = \sum_c \delta(c_t, c) f(c_t = c | d(t)) = f(c_t | d(t)) .$$

It means that instead of the Kronecker pdf of the pointer we should take $f(c_t | d(t))$ which is the estimated one. We denote $f(c_t | d(t)) = w_t \quad \text{(6.3)}$

and it is the weighting vector we use for estimation of parameters of individual components.

What remains is to show, how the pdf $f(c_t | d(t))$ can be obtained. And it is easy, because it holds

$$f(c_t | d(t)) \propto \int \int f(c_t, \alpha, \Theta | d(t)) \, d\alpha d\Theta$$

where $f(c_t, \alpha, \Theta | d(t))$ is the pdf all unknown objects (6.1). Using the decomposition, is reads

$$f(c_t | d(t)) \propto \int \int f(d_t | c_t, \psi_t, \Theta) f(c_t | \alpha) f(\alpha | d(t-1)) f(\Theta | d(t-1)) \, d\alpha d\Theta =$$

$$= \underbrace{\int f(d_t | c_t, \psi_t, \Theta) \, d\Theta}_{(A)} \int f(\alpha | d(t-1)) \, d\alpha \underbrace{\int f(\Theta | d(t-1)) \, d\Theta}_{(B)}$$

where the first integral $(A)$ represents prediction from the component $c_t$ and the second integral is a prediction from the pointer model.

The evaluation of the pdf $f(c_t | d(t))$ represents the classification part of the mixture estimation. In the estimation part, we update the statistics not with the full data item according to (6.2) - update the active component with full data, but each component is updated with its weight (6.3) according to how it corresponds to the data item.

Full derivation with all details can be found in [1]
6.4 Least square estimation

We have a regression model (here the first order one) and data sample \( \{y_t, u_t\}_{t=1}^T \) and we want to estimate parameters of the model. The procedure is as follows

\[
y_1 = a_1 y_0 + b u_1 + k + e_1 \\
y_2 = a_1 y_1 + b u_2 + k + e_2 \\
\ldots \\
y_t = a_1 y_{t-1} + b u_t + k + e_t
\]

\[ \rightarrow \]

\[
Y = X\theta + E
\]

where

\[
Y = \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_t \end{bmatrix}, \quad X = \begin{bmatrix} y_0 & u_1 & 1 \\ y_1 & u_2 & 1 \\ \vdots & \vdots & \vdots \\ y_{t-1} & u_t & 1 \end{bmatrix} = \begin{bmatrix} \psi'_1 \\ \psi'_2 \\ \vdots \\ \psi'_t \end{bmatrix}
\]

Then it holds (according to the minimum squares of errors \( e_t \))

\[
\hat{\theta}_t = (X'X)^{-1} X'Y \quad \text{estimate of reg. coef.}
\]

\[
\hat{Y} = X\hat{\theta}_t \quad \text{output prediction}
\]

\[
\hat{r}_t = \text{var}(Y - \hat{Y}) \quad \text{estimate of noise variance.}
\]

The procedure is illustrated in the following program

```matlab
// Estimation of scalar regression model of order 2 by LS
//
clc, clear, close // clear all
[u,t,n]=file (); // find working directory
chdir (dirname(n(1))); // set working directory ||

def('ps=genpsi(t,n,y,u)', 'ps=[y(t-(1:n)),u(t-(0:n)) 1]', 'c') // how the regression vector is organized in the simulation
// specifically ps = [y(t) y(t-1) u(t) u(t-1) u(t-2) 1]

load dataT11.dat Sim // loads simulation
yt=Sim.Cy.yt; // output
ut=Sim.Cy.ut; // input
ord=Sim.Cy.ord; // order
th=Sim.Cy.th; // simulated parameters
nd=length(yt); // data length
s=3:nd; // actual time
```
6.5 Estimation of model order

Here is the program solving model order estimation problem

```matlab
clc, clear, close, close
[u,t,n]=file();
chdir(dirname(n(1)));
getd(); mode(0)

// Data simulation
nd=100;
a1=.6; b0=1; sd=.01;
y=zeros(1,nd);
u=rand(1,nd,'n');
for t=2:nd
    y(t)=a1*y(t-1)+b0*u(t)+sd*rand(1,1,'n');
end

// Estimation
ord=5; // initial model order
// output
Y=y(ord+1:nd)';

// regression vector
x1=u(ord+1:nd)';
x2=y(ord:nd-1)'; // once delayed
x3=u(ord:nd-1)';
x4=y(ord-1:nd-2)'; // twice delayed
```
x5 = u(ord -1:nd -2)’;
x6 = y(ord -2:nd -3)’; // three times delayed
x7 = u(ord -2:nd -3)’;
x8 = y(ord -3:nd -4)’; // four times delayed
x9 = u(ord -3:nd -4)’;
x10 = y(ord -4:nd -5)’; // five times delayed
x11 = u(ord -4:nd -5)’;
x12 = ones(x1)’; // constant

X = [x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12];
// b0 a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 k
disp “b0 a1 b1 a2 b2 a3 b3 a4 b4 a5 b5 k”

th = inv(X’*X)*X’*Y // LS estimate of parameters
set (scf (), ’position’, [500 400 600 500])
bar(th)

6.6 Initialization of mixture estimation

Let us have static component (the component index is omitted)

\[ y_t = k + e_t. \]

Suppose, we have one prior regression vector data item \( y_0 \), constructed from the expert information. We want to initialize this component, i.e. to set the initial statistics \( V \) and \( \kappa \) so that, it corresponds to the expert information.

First of all, we have to realize, how the statistics are computed. The update is

\[
V_{\text{new}} = V_{\text{old}} + \Psi \Psi’ \\
\kappa_{\text{new}} = \kappa_{\text{old}} + 1
\]

where \( \Psi = [y_1, y_2, 1]’ \). If we have no old statistics, we set them zero. So the construction of the statistics reduces to

\[
V = \Psi \Psi’, \quad \kappa = 1.
\]

Remark: Notice, that \( \Psi \Psi’ \) carries the information about the system, while \( \kappa \) says, that the information was brought by a single data item.

Now, if we want to do the prior information stronger with respect to the several new measured data i.e. to extract it from more that one data item, we can add the expert data item several times - say \( m \)-times. Then we have

\[
V = m\Psi \Psi’ \quad \text{and} \quad \kappa = m.
\]

The information carried by \( V \) is still the same - it gives the parameters corresponding to the fictitious data item, but \( \kappa \) says, that the information comes as if from \( m \) data items, i.e. it is stronger in the sense that only \( m \) measured data can compete with is and to start to change the prior estimates. Thus, setting initial \( \kappa \) to some number and multiplying the initial information matrix by the same number means increasing the impact of prior setting of estimation with respect to coming measured data.
The following program illustrates the situation

```matlab
// Initialization of mixture estimation
// ----------------------------------------------------------
clc, clear, close // clear all
[u,t,n]=file(); // find working directory
chdir(dirname(n(1))); // set working directory
mode(0) // write values if no ;

load simple.dat
// expert knowledge
ps01=[8;1]+[-4;1]; // center of 1st component
ps02=[0;5]+[1;-3]; // center of 2nd component
nd=100; // numb. of values for estimation

// construction of information matrix
V=list(); thE=list(); cvE=list();
k=15; // numb. of prior data (as if)
ka=k*ones(1,2); // counter
V(1)=k*[ps01;1]*[ps01;1]'; // inf. mat. for 1st component
V(2)=k*[ps02;1]*[ps02;1]'; // inf. mat. for 2nd component

// point estimates of component parameters
thE(1)=ps01; // 8,1 in sim. initial
thE(2)=ps02; // 0,5 in sim. parameters
cvE(1)=.1*eye(2,2); // initial noise covariance
cvE(2)=.1*eye(2,2); // initial noise covariance

// pointer model statistics and parameter
nu=rand(2,1,'u'); // pointer-model statistics
al=nu/sum(nu); // pointer-model parameter

for t=1:nd // time loop ================
    // computation of weights
    for j=1:2
        [xxx,mL(j)]=GaussN(yt(:,t),thE(j),cvE(j)); // proximity
    end
    mp=mL-max(mL); // normalization
    m=exp(mp); // exponent
    w=m.*al; w=w/sum(w); // component weights
    wt(:,t)=w; // store

    // recomputation of statistics
    Ps=[yt(:,t);1];
    for j=1:2
        V(j)=V(j)+w(j)*Ps*Ps';
        ka(j)=ka(j)+w(j);
    end

    // point estimates
    thE(j)=(inv(V(j)(3,3))*V(j)(3,1:2))';
    if t>50
        // in the beginning, cvE is fixed
```

21
cvE(j)=(V(j)(1:2,1:2)-thE(j)*V(j)(3,1:2))/ka(j);
end
end
nu=nu+w;        // pointer-model statistics
al=nu/sum(nu);  // and parameter

th1(:,t)=thE(1);    // stor for
th2(:,t)=thE(2);    // plot
end
[xxx,ce]=max(wt,'r'); ce=ce';

// Results
wrong=nd-sum(ct(1:nd)==ce(1:nd))
from=nd

set(scf(2),'position',[900 100 800 600])
subplot(211),plot(th1')
subplot(212),plot(th2')

References