Elementary Scilab programs

Contents

1 Simulation and estimation

1.1 Binary model

```
// model_1.sce
// Estimation of binary model
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
p=.3;                    // model probability
y=(rand(1,nd,'u')>p)+1;  // data

// estimation
n=zeros(1,2);            // initial statistcs
for t=1:nd
  n(y(t))=n(y(t))+1;     // statistics update
end
pE=n(1)/sum(n)           // point estimates
```

1.2 Categorical model

```
// model_2.sce
// Estimation of categorical model
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
p=[.3 .1 .6];            // model parameters
for t=1:nd
  y(t)=sum(cumsum(p)<rand(1,1,'u'))+1; // data
end

// estimation
n=zeros(1,length(p));    // initial statistics
for t=1:nd
  n(y(t))=n(y(t))+1;     // statistics update
end
pE=n/sum(n)              // point estimates
```

1.3 Binomial model

```
// model_3.sce
// Estimation of binomial model
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
p=.3;                    // model parameter
N=5;                     // maximum value of y
for t=1:nd
  y(t)=sum(rand(1,N,'u')<p); // data
end

// estimation
S=0;                     // intial
ka=0;                    //   statistics
for t=1:nd
```

```
  S=S+y(t);              // update of
  ka=ka+1;               //    statistics
end
pE=S/(N*ka)              // point estimates
```

1.4 Poisson model

```
// model_4.sce
// Esimation of Poisson model
// ------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
lam=3;                   // model parameter
N=100;                   // length of binomial experiment
p=lam/N;                 //   for generation of Poisson
for t=1:nd
  y(t)=sum(rand(1,N,'u')<p); // data
end

// estimation
S=0;                     // initial
ka=0;                    //   statistics
for t=1:nd
  S=S+y(t);              // update of
  ka=ka+1;               //   statistics
end
lamE=S/ka                // point estimates
```

1.5 Constant regression model

```
// model_5.sce
// Estimaion of constant regression model
// ------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
```

```
m=5;                      // expectation
r=2;                      // variance
y=m+sqrt(r)*rand(1,nd,'n'); // data

// estimation
mE=mean(y)                // point
rE=variance(y)            //   estimates
```

1.6 Explanatory regression model

```
// model_6.sce
// Esimation of explanatory regression model
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                   // number of data
c1=2; c2=-3;              // regression coefficients
r=.2;                     // variance
x1=5*rand(1,nd,'n');      // first explanatory variable
x2=ceil(3*rand(1,nd,'u')); // second explanatory variable
y=c1*x1+c2*x2+sqrt(r)*rand(1,nd,'n'); // output

// estimation
for t=2:nd
  Y(t,1)=y(t);            // computation of Y
  X(t,:)=[x1(t) x2(t)];   // computation of X
end
cE=inv(X'*X)*X'*Y         // esimate of parametrs
yp=X*cE;                  // prediction
ep=y'-yp;                 // prediction error
rE=variance(ep)           // estimate of variance
```

1.7 Dynamic regression model

```
// model_7.sce
// Estimation of dynamic regression model
// -------------------------------------
```

```
clc,clear,close,mode(0)

//simulation
nd=200;                         // number of data
a1=.6; a2=-.3; b0=1; k=1;       // regression coefficients
r=.2;                           // variance
u=5*sin(2*%pi*(1:nd)/nd)+1;     // input
y(1)=2; y(2)=-1;                // initial conditions
for t=3:nd
  y(t)=a1*y(t-1)+a2*y(t-2)+b0*u(t)+k+sqrt(r)*rand(1,1,'n');
end                             // output

// estimation
V=zeros(5,5);                   // initial
ka=0                            //   statistics
for t=3:nd
  Ps=[y(t) y(t-1) y(t-2) u(t) 1]'; // extended reg. vector
  V=V+Ps*Ps';                   //update of
  ka=ka+1;                      // statistics
end
Vy=V(1,1);
Vyp=V(2:$,1);
Vp=V(2:$,2:$);
thE=inv(Vp)*Vyp                 // pont estimates of rerg. coef.
```

1.8 Exponential model

```
// model_8.sce
// Estimation of exponential model
// ----------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                    // number of data
a=.3;                      // model parametr
for t=1:nd
  y(t)=-log(rand(1,1,'u'))/a; // data
end

// estimation
```

```
aE=1/mean(y)             // point estimates
```

1.9 Uniform model

```
// model_9.sce
// Estimation of uniform model
// ------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                  // number of data
L=3; U=5;                // parametrs
for t=1:nd
  y(t)=L+(U-L)*rand(1,1,'u'); // data
end

// estimation
LE=%inf;                 // initial
UE=-%inf;                //   statistics (parameters)
for t=1:nd
  if y(t)<LE, LE=y(t); end // statistics
  if y(t)>UE, UE=y(t); end //   update
end
LE,UE                    // parameter estimates
```

2 Initialization

2.1 Binary model

```
// init_1.sce
// Initialization of coin estimation
// ------------------------------------
clc, clear, close, mode(0);

// simulation
nd=500;                  // number of data
p=.7                     // model parameter
y=(rand(1,nd,'u')>p)+1;  // data
```

```
// initialization
ka=1;                    // strength of initialization (counter)
pE=.5;                   // initial parameter  p=P(y=1)
S=ka*[pE 1-pE];          // statistics  [numb. of 1, numb. of 2]

// estimation
for t=1:nd
  S(y(t))=S(y(t))+1;     // statistics
  ka=ka+1;               //   update
  pE=[pE S(1)/ka];       // estimates
end

// result
plot(0:nd,pE)
```

2.2 Categorical model

```
// init_2.sce
// Initialization of categorical model estimation
// - model f(y|x), y=1,2; x=1,2,3
// -------------------------------------
clc, clear, close, mode(0);

// simulation
nd=500;                  // number of data
px=[.3 .5 .2];           // parameter for x-model
py=[.2 .7 .4             // parameter for y-model
    .8 .3 .6];
for t=1:nd
  x(t)=sum(cumsum(px)<rand(1,1,'u'))+1;           // var. x
  y(t)=sum(cumsum(py(:,x(t)))<rand(1,1,'u'))+1;   // var. y
end

// initialization
ka=1;                    // counter
pE=[.3 .7 .2             // initial parametr
    .7 .3 .8];
V=pE*ka;                 // statistics
pEt=pE(1,1);
```

```
// estimation
for t=1:nd
  V(y(t),x(t))=V(y(t),x(t))+1;     // statistics
  ka=ka+1;                         //   update
  for j=1:max(x)
    pE(:,j)=V(:,j)/sum(V(:,j));    // estimates
  end
  pEt=[pEt pE(1,1)];
end

// result
set(scf(),'position',[500 200 500 400])
plot(0:nd,pEt)

disp('simulated parameters')
disp(py)
disp('estimated parameters')
disp(pE)
```

2.3 Regression model

```
// init_3.sce
// Initialization of regression model estimation
// - model y = c1.x1 + c2.x2 + c3.x3 + k + e
// - x = m + sd.v (v white noise)
// ----------------------------------------
clc, clear, close, mode(0);

// simulation
nd=500;                 // number of data
m=[2 -1 3]';            // parameters for
sdx=[.5 .1 .3           //   x-model
      0 .2 .1
      0  0 .8];
c=[4 -2 2];             // parameters for
k=5;                    //   y-model
sdy=.7;
for t=1:nd
  x(:,t)=m+sdx*rand(3,1,'n');           // data x
  y(t)=c*x(:,t)+k+sdy*rand(1,1,'n');    // data y
```

```
end

// initialization
// |  c  |k|
thE=[5 0 1 3]';          // initial parametrs
ka=1;                    // initial counter
V=eye(5,5);
V(2:$,1)=thE;
V(1,2:$)=thE';
V=V*ka;                  // statistics (rank thE + 1)
tht=thE;

// estimation
for t=1:nd
  Ps=[y(t) x(:,t)' 1]';  // extended reg. vector
  V=V+Ps*Ps';            // statistics
  ka=ka+1;               //   update
  Vy=V(1,1);
  Vyp=V(2:$,1);
  Vp=V(2:$,2:$);
  thE=inv(Vp)*Vyp;       // estimates
  tht=[tht thE];
end

// results
set(scf(),'position',[500 200 500 400])
plot(0:nd,tht)
title 'Evolution of the estimated parameters'
legend('c1','c2','c3','k');

disp('simulated parameters')
disp([c'; k])
disp('initial parameters')
disp(tht(:,1))
disp('estimated parameters')
disp(thE)
```

3 Prediction

3.1 Zero-step prediction

3.1.1 Regression model

```
// pred_1.sce
// Prediction with regression model
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                      // number of data
c=[5 3 -1];                  // regression coefficients
sd=1.2;                      // standard deviation
x=[2;-1;3]*ones(1,nd)+.5*(rand(3,nd,'n')); // data x
y=c*x+sd*rand(1,nd,'n');                   // data y

select 1  // SELECT type of prediction: 1-point, 2-simulated
case 1              // point prediction
for t=1:nd
  yp(t)=c*x(:,t);
end
case 2              // simulated prediction
for t=1:nd
  yp(t)=c*x(:,t)+sd*rand(1,1,'n');
end
end

// resulrs
set(scf(),'position',[500 200 500 400])
plot(1:nd,y,1:nd,yp)
legend('y','yp');
disp('Relative prediction error')
disp(variance(y-yp')/variance(y))
```

3.1.2 Categorical model

```
// pred_2.sce
// Prediction with categorical model
```

```
// -------------------------------------
clc,clear,close,mode(0)

// simulation
nd=200;                       // number of data
px=[.3 .2 .3 .2];             // pars for model x
py=[.8 .1 .1 .3               // pars for model y
    .1 .8 .1 .5
    .1 .1 .8 .2];
for t=1:nd
  x(t)=sum(cumsum(px)<rand(1,1,'u'))+1;         // data x
  y(t)=sum(cumsum(py(:,x(t)))<rand(1,1,'u'))+1; // data y
end

select 2       // SELECT type of prediction
case 1              // point prediction
for t=1:nd
  [nill,yp(t)]=max(py(:,x(t)));
end
case 2              // simulated prediction
for t=1:nd
  yp(t)=sum(cumsum(py(:,x(t)))<rand(1,1,'u'))+1;
end
end

// results
set(scf(),'position',[500 200 500 400])
plot(1:nd,y,'o',1:nd,yp,'x')
legend('y','yp');
disp('Accuracy')
disp(sum(y==yp)/nd)
```

3.2 K-step prediction

3.2.1 Regression model with known parameters

```
// pred_3.sce
// NP-STEP PREDICTION WITH CONTINUOUS MODEL (KNOWN PARAMETERS)
// Experiments
// Change: - np = number of steps of prediction
```

```
//         - r  = noise variance
//         - th = model parametrs
//         - u  = input signal
// -------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=100;                             // number of data
np=5;                               // length of prediction (np>=1)
//  b0 a1 b1  a2 b2 k
th=[1 .4 -.3 -.5 .1 1]';            // regression coefficients
r=.02;                              // noise variance
u=sin(4*%pi*(1:nd)/nd)+rand(1,nd,'norm'); // input
y(1)=1; y(2)=-1;                    // prior data


// TIME LOOP
for t=3:(nd-np)                     // time loop (on-line tasks)
  // prediction
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]'; // first reg. vec for prediction
  yy=ps'*th;                        // zero prediction for time = t (np=0)
  for j=1:np                        // loop of predictions for t+1,...,t+np
    tj=t+j;                         // future times for prediction
    ps=[u(tj); yy; ps(1:$-3); 1];   // reg.vecs with predicted outputs
    yy=ps'*th;                      // new prediction (partial)
  end
  yp(t+np)=yy;                      // final prediction for time t+np

  // simulation
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]';  // regression vector for sim.
  y(t)=ps'*th+sqrt(r)*rand(1,1,'norm');      // output generation
end

// Results
s=(np+3):(nd-np);
scf(1);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[1 nd -3 5])
legend('output','prediction');
title(string(np)+'-steps ahead prediction')
```

```
RPE=variance(y(s)-yp(s))/variance(y)    // relative prediction error
```

3.2.2 Regression model with unknown parameters

```
// pred_4.sce
// N-STEP PREDICTION WITH CONTINUOUS MODEL (WITH ESTIMATION)
// Experiments
// Change: - np = number of steps of prediction
//         - r  = noise variance (effect on estimation)
//         - th = model parametrs
//         - u  = input signal (effect on estimation)
// ------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=100;                             // number of data
np=5;                               // length of prediction (np>=1)
nz=3;                               // starting time (ord+1)
//  b0 a1 b1  a2 b2 k
th=[1 .4 -.3 -.5 .1 1]';            // regression coefficients
r=.2;                               // noise variance
u=sin(4*%pi*(1:nd)/nd)+rand(1,nd,'norm'); // input
y(1)=1; y(2)=-1;                    // prior data
Eth=rand(6,1,'n');                  // prior parametrs

nu=zeros(4,2);
for t=nz:(nd-np)                    // time loop (on-line tasks)
  // prediction
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]'; // regression vector
  yy=ps'*Eth;                       // first prediction at t+1
  for j=1:np                        // loop of predictions for t+2,..,t+np
    tj=t+j;                         // future times for prediction
    ps=[u(tj); yy; ps(1:$-3); 1];  // reg.vecs with predicted outputs
    yy=ps'*Eth;                     // new prediction (partial)
  end
  yp(t+np)=yy;                      // final prediction for time t+np

  // simulation
  ps=[u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]';  // regression vector for sim.
  y(t)=ps'*th+sqrt(r)*rand(1,1,'norm');      // output generation
```

```
  // estimation
  Ps=[y(t) u(t) y(t-1) u(t-1) y(t-2) u(t-2) 1]';  // reg.vect. for estim.
  if t==nz, V=1e-8*eye(length(Ps)); end // initial information matrix
  V=V+Ps*Ps';                           // update of statistics
  Vp=V(2:$,2:$);
  Vyp=V(2:$,1);
  Eth=inv(Vp+1e-8*eye(Vp))*Vyp;         // point estimates
  Et(:,t)=Eth(:,1);                     // stor est. parameters
end

// Results
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)

set(scf(1),'position',[100 100 1200 400]);
subplot(121),plot(Et')
set(gca(),"data_bounds",[0 nd+1 -1 2])
title('Evolution of estimated parameters')
subplot(122)
s=(np+3):(nd-np);
plot(s,y(s),'.:',s,yp(s),'rx')
set(gca(),"data_bounds",[1 nd -3 5])
legend('output','prediction');
title([string(np),'-steps ahead prediction'])
```

3.2.3 Categorical model with known parameters

```
// pred_5.sce
// PREDICTION WITH DISCRETE MODEL (OFF-LINE)
// Experiments
// Change: - np  = number of steps of prediction
//         - th1 = model parametrs
//         - u   = input signal (effect on estimation)
//         - uncertainty of the system (effect on estimation)
// ----------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=50;                            // length of data sample
```

```
np=0;                                // length of prediction (np>=1)
th1=[0.98 0.01 0.04 0.97]';          // parameters for simulation (for y=1)
th=[th1 1-th1];                      // all parameters
u=(rand(1,nd)>.3)+1;                 // input
y(1)=1;

// SIMULATION
for t=2:nd
  i=2*(u(t)-1)+y(t-1);               // row of the table
  y(t)=(rand(1,1,'u')>th(i,1))+1;    // output generation
end

// PREDICTION
yy=ones(1,nd);                       // fictitious predicted output
for t=2:(nd-np)
  i=2*(u(t)-1)+y(t-1);               // row of the table
  yy=(rand(1,1,'u')>th(i,1))+1;      // prediction generation
  for j=1:np
    i=2*(u(t+j)-1)+yy;               // row of the table
    yy=(rand(1,1,'u')>th(i,1))+1;    // prediction generation
  end
  yp(t+np)=yy;                       // np-step predction
end

// Results
disp(th,' Model parameters'), disp(' ')

s=(np+3):nd;
plot(s,y(s),'.:',s,yp(s),'rx')
set(gcf(),'position',[600 100 800 400])
set(gca(),"data_bounds",[0 nd+1 .9 2.1])
legend('output','prediction');
title(string(np)+'-steps ahead prediction')

Wrong=sum(y(:)~=yp(:)), From=nd
```

3.2.4 Categorical model with unknown parameters

```
// pred_6.sce
// PREDICTION WITH DISCRETE MODEL (ON-LINE)
```

```
// Change: - length of prediction
//         - uncertainty of the simulated model
//         - imput signal
//         - study the beginning when estimation is not finished
//           how can we secure quicker transient phase of estimation?
// Remark: another way og generation is
//            y(t)=sum(rand(1,1,'u')>cumsum(th(i,:)))+1;
// ---------------------------------------------------------------------
exec("ScIntro.sce",-1),mode(0)

nd=150;                            // number of data
np=2;                              // length of prediction
th1=[0.98 0.01 0.04 0.97]';        // parameters (for y=1)
th=[th1 1-th1];                    // all parameters
u=(rand(1,nd+np,'u')>.3)+1;        // input
y(1)=1;

// TIME LOOP
nu=1e-8*ones(4,2);
Et=zeros(4,nd-np);
for t=2:nd                         // time loop
  // prediction
  i=2*(u(t)-1)+y(t-1);             // row of the table
  yy=(rand(1,1,'u')>th(i,1))+1;    // prediction generation
  for j=1:np
    i=2*(u(t+j)-1)+yy;             // row of the table
    yy=(rand(1,1,'u')>th(i,1))+1;  // prediction generation
  end
  yp(t+np)=yy;                     // np-step predction

  // simulation
  i=2*(u(t)-1)+y(t-1);
  y(t)=(rand(1,1,'u')>th(i,1))+1;

  // estimation
  i=2*(u(t)-1)+y(t-1);             // row of model matrix
  nu(i,y(t))=nu(i,y(t))+1;         // statistics update
  Eth=nu./(sum(nu,2)*ones(1,2));   // pt estimates
  Et(:,t)=Eth(:,1);
end
```

```
// Results
disp(' Simulated parameters')
disp(th)
disp(' Estimated parameters')
disp(Eth)

s=np+2:np+51;
set(scf(),'position',[100 100 1000 400])
subplot(121),plot(Et')
title('Evolution of estimated parameters')
set(gca(),"data_bounds",[0 nd-np+1 -.1 1.1])
subplot(122),plot(s,y(s),s,yp(s),'.:')
title('First 50 outputs and their prediction')
set(gca(),"data_bounds",[s(1) s($) .9 2.1])

s=np+2:nd;
Wrong=sum(y(s)~=yp(s))
From=nd-np
```

4 Classification

4.1 Known components

```
// class_1.sce
// Classification with regression components
// - known models of components
// ------------------------------------
clc,clear,close,getd(),mode(0)

// simulation
nd=2000;                 // number of data
p=[.2 .5 .3];            // pointer parametr
thS=[1 5 3               // model parametrs
     1 2 8];             //   three clusters: [1 1],[5 2],[3 8]
sd=1.2;                  // standard deviation
for t=1:nd
  c(t)=sum(cumsum(p)<rand(1,1,'u'))+1;            // pointer
  x(:,t)=thS(:,c(t))+sd*eye(2,2)*rand(2,1,'n');   // data
```

```
end
nc=size(thS,2);          // number of components

// classification
for t=1:nd
  for j=1:nc
    q(j)=Gauss(x(:,t),thS(:,j),sd^2*eye(2,2)); // proximity
  end
  fc=q/sum(q);           // pointer distribution
  [nill,cp(t)]=max(fc);  // pointer value
  wt(:,t)=fc;
end

// result
Accuracy=sum(c==cp)/nd
```

4.2 Teacher

```
// class_3.sce
// Classification in continuous data space
// - learning with a teacher
// - recursive
// -------------------------------------
clc,clear,close,getd(),mode(0)
getd c:\functions        // library of functions

// simulation
nL=500;                  // number of data for learning
nT=200;                  // number of data for testing
p=[.2 .5 .3];            // pointer model parameters
thS=[1 5 3               // three clusters: [1 1],[5 2],[3 8]
     1 2 8];             // two variables (comp. pars)
for t=1:(nL+nT)
  y(t)=sum(cumsum(p)<rand(1,1,'u'))+1;          // data y
  x(:,t)=thS(:,y(t))+.8*eye(2,2)*rand(2,1,'n'); // data x
end
[nv,nc]=size(thS);       // numb. of variables and components

// estimation
xL=x(:,1:nL);            // data for
```

```
yT=y(1:nL);               //   learning
m=thS+.5*rand(nv,nc);     // initial parameters
ka=1*ones(1,nc);          // initial counter
S=m.*(ones(nv,1)*ka);     // initial statistics
for t=1:nL
  for j=1:nc
    q(j)=GaussN(xL(:,t),m(:,j),.1); // proximity
  end
  w=q/sum(q);             // weights
  wt(:,t)=w;
  for j=1:nc
    S(:,j)=S(:,j)+w(j)*xL(:,t); // statistics
    ka(j)=ka(j)+w(j);           //   update
    m(:,j)=S(:,j)/ka(j); // parameter estimates
  end
end

// classification
xT=x(:,nL+(1:nT));        // data for
yT=y(nL+(1:nT));          //   testing
for t=1:nT
  for j=1:nc
    q(j)=GaussN(xT(:,t),m(:,j),.1); // proximity
  end
  fy=q/sum(q);            // prediction of the pointer
  [nill,yp(t)]=max(fy);   // value of the pointer
  wt(:,t)=fy;
end

// result
Accuracy=sum(yT==yp)/nT
```

4.3 Naive Bayes

```
// class_4.sce
// Classification in continuous data space
// - naive Bayes with teacher
// ----------------------------------------
clc,clear,close,getd(),mode(0)
getd c:\functions
```

```
// simulation
nL=500;             // number of data for learning
nT=200;             // number of data for testing
p=[.2 .5 .3];       // parameters for pointer model
thS=[
1 5 3               // parameters for static Gaussian components
1 2 8               // -  three clusters, five variables
2 9 5
8 1 3
1 9 4];
[nv,nc]=size(thS);  // number of variables and comonents
for t=1:(nL+nT)
  y(t)=sum(cumsum(p)<rand(1,1,'u'))+1;       // target (pointer)
  for i=1:nv
    x(i,t)=thS(i,y(t))+.2*rand(1,1,'n');     // variables x
  end
end

// estimation with teacher (known components)
xL=x(:,1:nL);                       // data for
yT=y(1:nL);                         //   learning
S=.01*ones(nv,nc);                  // initial S
ka=.01*ones(nv,nc);                 // initial kappa
for t=1:nL
  j=yT(t);                          // components from teacher
  for i=1:nv
      S(i,j)=S(i,j)+xL(i,t);        // update of S (sum)
      ka(i,j)=ka(i,j)+1;            //        of ka (count)
      m(i,j)=S(i,j)/ka(i,j);        // parameters
  end
end

// classification
xT=x(:,nL+(1:nT));                  // data for
yT=y(nL+(1:nT));                    //   testing
for t=1:nT
  qi=1;
  for i=1:nv
    for j=1:nc
```

```
      q(j)=GaussN(xT(i,t),m(i,j),.1);   // prox. for i-th variable
    end                                 // - q propto f(xi|y)=[f1(xi|y),f2(xi|y)...]
    qi=qi.*q;                           // product for variables
  end                                   // - fy propto Prod(f(xi|y))
  fy=q/sum(q);                          // normalization
  [nill,yp(t)]=max(fy);                 // argument maxima = classification
  wt(:,t)=fy;
end

// result
Accuracy=sum(yT==yp)/nT                 // num.of positive/num.of all
```

4.4 Kernel estimation

```
// class_5b.sce
// Naive Bayes
// - kernel estimation
// - two dimensional normal y
// - comparison of various types of kernels
// ---------------------------------------
exec('SCIHOME/ScIntro.sce',-1); mode(0);

// data
est=1;    // <-- new estimation 1-yes, 0-no
ik=3;     // <-- select type of kernel
if est
  nd=200;
  th=[2 8
      5 1];
  sd{1}=[ 1 .5
        -.1  2];
  sd{2}=[ 2 -.5
         .2  1];
  al=[.6 .4];
  for t=1:nd
    c(t)=sampCat(al);
    y(:,t)=th(:,c(t))+uut(sd{c(t)})*randn(2,1);
  end
  save yy.dat y c nd
else
```

```
  load yy.dat y c nd
  nL=max(size(y));
end

// classification
select ik
  case 0, kr='kerfx';   disp Gauss
  case 1, kr='kerfx1';  disp Epanechnikov
  case 2, kr='kerfx2';  disp Biweight
  case 3, kr='kerfx3';  disp Triweight
  end

y1=y(:,find(c==1));                   // y in class 1
y2=y(:,find(c==2));                   // y in class 2
fc(1)=length(y1)/nd;                  // f(c=1)
fc(2)=length(y2)/nd;                  // f(c=2)
for i=1:2
  r1(i)=variance(y1(i,:));  // varince for kernel 1
  r2(i)=variance(y2(i,:));  // varince for kernel 2
end

for t=1:nd                             // loop for class.
  q=ones(2,1);
  for i=1:2
    execstr('q(1)=q(1)*'+kr+'(y1(i,:),y(i,t),r1(i));')   // proximity  f(c|y1)
    execstr('q(2)=q(2)*'+kr+'(y2(i,:),y(i,t),r2(i));')   // proximity  f(c|y1)
  end
  wp=q.*fc;
  w=wp/sum(wp);                        // weight
  wt(:,t)=w;
  cp(t)=amax(w,'r');                   // classif.
end

// results
space
ACC=acc(c,cp)

set(scf(),'position',[500 200 800 300]);
// hist and ker of y1
[f,s]=histc(y1,20,'b',0);
```

```
g=s(2)-s(1);
p=f./(sum(f)*g);
subplot(121)
bar(s,p,'c')
[z,x]=kerf(y1,.1,ik);
plot(x,z)

// hist and ker of y2
[f,s]=histc(y2,20,'b',0);
g=s(2)-s(1);
p=f./(sum(f)*g);
subplot(122)
bar(s,p,'c')
[z,x]=kerf(y2,.1,ik);
plot(x,z)
```

4.5 Mixture estimation

```
// class_6a.sce
// Classification in continuous data space
// - naive Bayes without teacher (= mixture estimation)
// -------------------------------------
clc,clear,close,getd(),mode(0)
getd c:\functions

// simulation
nI=20;              // number of initial data
nL=200;             // number of data for learninf
nT=200;             //number of data for testing
p=[.2 .5 .3];       // pointer parameters
thS=[               // model parameters
1 5 3               //   three clusters, five variables
1 2 8
2 9 5
8 1 3
1 9 4];
[nv,nc]=size(thS);
for t=1:(nI+nL+nT)
  y(t)=sum(cumsum(p)<rand(1,1,'u'))+1;  // targer data
  for i=1:nv
```

```
    x(i,t)=thS(i,y(t))+.2*rand(1,1,'n');// explanatory data
  end
end

// initiation
xI=x(:,1:nI);       // data for
yI=y(1:nI);         //   init.
ka=1*ones(nv,nc);   // counter
for j=1:nc
  s=find(yI==j);
  for i=1:nv
    m(i,j)=mean(xI(i,s));          // component expecatations
    r(i,j)=variance(xI(i,s));      // component variances
    S(i,j)=ka(i,j)*m(i,j);         // statistics
  end
end

// estimation
xL=x(:,nI+(1:nL));  // data for
yL=y(nI+(1:nL));    //   learning
for t=1:nL
  for i=1:nv

    for j=1:nc
      [nill,Lq(j)]=GaussN(xL(i,t),m(i,j),.1); // proximity
    end                                       // in logarithm
    Lq=Lq-max(Lq);  // pre-normaliazation
    q=exp(Lq);      // log --> value
    w=q/sum(q);     // weight
    for j=1:nc
      S(i,j)=S(i,j)+w(j)*xL(i,t);  // statistic
      ka(i,j)=ka(i,j)+w(j);        //   update
      m(i,j)=S(i,j)/ka(i,j);       // estimate
    end

  end
end

// classification
xT=x(:,nI+nL+(1:nT));    // data for
```

```
yT=y(nI+nL+(1:nT));      //   classification
for t=1:nT
  for i=1:nv
    Lp=0;
    for j=1:nc
      [nill,Lq(j)]=GaussN(xT(i,t),m(i,j),.1); // proximity
    end
    Lq=Lq-max(Lq);
    Lp=Lp+Lq;            // sum in log = multiplication
  end
  q=exp(Lp);
  fy=q/sum(q);           // pointer prediction (= weights)
  [nill,yp(t)]=max(fy);  // pointe value
  wt(:,t)=fy;
end

// result
Accuracy=sum(yT==yp)/nT
```

4.6 C-means algorithm

```
// C means algorithm
// ---------------------------------------------
exec SCIHOME/ScIntro.sce, mode(0)

m=2;                                    // fuzzy coefficient
x=[1.2 2.5 6.5 7.8 9.3]                 // data
c=[4 5]'                               // initial centers

for ite=1:50                            // loop of iterations
  printf('Iteration -- %d\n',ite)

// recomputation of weights
  for i=1:2
    for j=1:5
      s=0;
      for k=1:2
        s=s+(abs(x(j)-c(i)))**(m/(m-1)); // membership function
      end
      u(i,j)=1/s;
```

```
    end
  end
  u=u./(ones(2,1)*sum(u,1));            // normalization

  // construction of centers
  cOld=c;
  c=(u*x')./sum(u,2)

  if sum(abs(c-cOld))<.01               // test for end
    break
  end
end
printf('\n')
cFinal=c
```

5 Functions

5.1 Gaussian pdf

```
function [p,Lp]=GaussN(x,m,R)
  // [p 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)));  //pause
  ex=(x-m)'*inv(R+1e-8*eye(n,n))*(x-m);
  Lp=Lp-.5*ex;
  p=exp(Lp);
//  pause
endfunction
```

5.2 Histogram for continuous data

```
function [f,sm]=histc(x,n,c,r)
  // histogram of x (continuous)
  // x    data
  // n    number of columns
  // c    color
  // r \in (0,1) - width of the columns
  if argn(2)<4, r=.8; end
  if argn(2)<3, c='b'; end
  if argn(2)<2, n=20; end
  minx=min(x);
  maxx=max(x);
  h=(maxx-minx)/(n);
  s=minx:h:maxx;
  for i=1:n
    f(i)=length(find((x>=s(i))&(x<s(i+1))));
  end
  k=find(x==s(n));
  if ~isempty(k)
    f(n)=f(n)+length(k);
  end
  for i=1:n
    sm(i)=(s(i)+s(i+1))/2;
  end
  if r>0, bar(sm,f,r,c); end
endfunction
```

5.3 Histogram for discrete data

```
function v=histd(x,s,r)
  // v    histogram of x (discrete)
  //      with interupted values on x axis
  // x    data
  // s    all points on axis x (incl. zeros before and after)
  // r \in (0,1) - width of the columns
  if argn(2)<3, r=.8; end
  if argn(2)<2
    vx=vals(x);
    s=vx(1,:);
```

```
  end
  minx=min(s);
  maxx=max(s);
  v=vals(x);
  s=minx:maxx;
  h=zeros(s);
  h(v(1,:)-minx+1)=v(2,:);
  bar(s,h,r)
endfunction
```

5.4 Kernel function

```
function [z,xx]=kerf(y,h,ik)
  // Gaussian kernel density of data y
  // r    variance
  // h    step
  if argn(2)<2, n=20; end
  if argn(2)<3, ik=0; end

  mi=min(y);
  ma=max(y);
  s=mi:h:ma;
  xx=min(s):h:max(s);
  r=variance(y);
  k=0;
  for x=xx
    k=k+1;
    z(k)=0;
    select ik
    case 0, z(k)=z(k)+kerfx(y,x,r);
    case 1, z(k)=z(k)+kerfx1(y,x,r);
    case 2, z(k)=z(k)+kerfx2(y,x,r);
    case 3, z(k)=z(k)+kerfx3(y,x,r);
    end
  end
endfunction
```