Mathematical methods of data analysis

Ivan Nagy

Dept. of Applied Mathematics, FTS, CTU-Prague

Contents

Lecture 1

1 Naive Bayes classification

1.1 Preliminaries

Laboratory

On the web

https://www.fd.cvut.cz/personal/nagyivan/WebLabNew/MAIN.pdf

you can find an overview of Stochastic Systems under Bayesian methodology. It begins with the most frequent stochastic **models** each of them supplied with the formulas to their estimation. Then a treatment of the **initialization** necessary for good estimation is discussed. After it the task of **prediction** (estimation of the future values of the modeled variable) is introduced. The the task of **filtration** (estimation of the values of an unmeasured variable) follows. Finally the problem of **classification** (sorting values of the measured variable into several classes) based on mixture estimation is tackled. An extension to the theory of mixture estimation are so called metamixtures. Here, a solution of the problem of classification of a multivariate discrete variable is indicated in the form of marginal mixtures.

Distribution

Let us have normal distribution $f(y,\mu) = N_y(\mu,1)$. This distribution generates data around the value μ (expectation). The closer the generated value lies to μ, the better it agrees with the model (given by the expectation μ) and the grater is the value of the distribution. The value $\tilde{y}$ for which is $f(\tilde{y}|\Theta)$ very small hardly belongs to this distribution.

Model and its estimation

For y being random variable, the description is a distribution

$$f(y|\Theta) \sim \text{distribution}$$

where Θ - parameter (represents the reality), y - generated variable (data). For more see the Laboratory.

The reality Θ is action from which we have results y. The model says what actions we can probably expect from given reality, i.e.

$$f(\text{result}|\text{action}).$$

However, we also can recognize the action from the measured results. It is estimation with the model

$$f(\Theta|y) \propto f(y|\Theta) f(\Theta).$$

For given data $y_1, y_2, \cdots y_T$ we can look for parameter which fits them best, i.e. which maximizes the distribution (see the previous paragraph) and its factorization

$$f(y_1, y_2, \cdots y_T, \Theta) = f(\Theta|y_1, y_2, \cdots y_T) f(\Theta) = f(\Theta) \prod_{t=1}^{T} f(y_t|\Theta),$$

where the last adjustment of the expression is due to the independency of measured variables. Here, $\prod_{t=1}^{T} f(y_t|\Theta) = L_T(\Theta)$ is likelihood (reflecting information brought by data) and $f(\Theta)$ is prior (reflecting the expert knowledge). The maximization leads to a definition of suitable statistics which can be recursively updated wit coming data. The parameter estimates can be derived from the updated statistics. The for of the statistics, their update and the construction of the estimates for the most frequent model is described in the Laboratory, Section 2 Models and their estimation.

Example: Normal model $f(y|\mu)$ generated by the equation $y_t = \mu + e_t$.

The statistic are $S_t = \sum_{\tau=1}^{t} y_\tau$ and $\kappa_t = t$.

Update

$$S_t = S_{t-1} + y_t; \quad \kappa_t = \kappa_{t-1} + 1$$

Estimate

$$\hat{\mu}_t = \frac{S_t}{\kappa_t}.$$

1.2 Introduction

We will mainly deal with classification. A general formulation of this task as follows: We have a multimodal system - it acts in several different working regimes and thus produces several different types of data.

Example: *Traffic intensities: morning are high, at noon a bit lower, afternoon again high, evening lover, at night practically zero.* □

On this system, we measure the variable y and want to sort its measured values to the classes corresponding with the working regimes.

Remark: If the activity of the working classes is known, the task is trivial. An interesting case is, when the switching of the classes is not known or even if the classes themselves are not known.

In the case, when the switching of working regimes is not known, we must learn about the system. The learning can be:

1. **With a teacher** - here we have some learning data at disposal where both the values of y and also the number of working regime c. From these data we train the model. After this, we measure only values of y and guess the values of c - the corresponding regime.
2. **Without a teacher** - here no training data are at disposal and we must determine the classes (clustering) and to assign them by the regimes (classification).

1.3 Model generally

The set of classifiers, we will be mainly interested in, is based on a model of the variables of the system under investigation. The variables can be continuous (with real values) or discrete (with a finite number of values).

Example *1: The time of traveling from the point A to B.*

$$y = ax$$

where $a = \frac{1}{v}$, v is the average speed of traveling and $x = s$, s is the distance between the points A and B. □

Example *2: The same as previous but with uncertainty in speed (with various drivers).*

$$y = ax + e$$

where e is random variable representing uncertain disturbance. □

Remark: Notice, that e is random variable and due to it y is also a random variable. So, y is no longer determined by its value but by its distribution - it is by its all possible values and their "probabilities".

Example *3: The same as previous but using distribution.*

$$f(y|x)$$

where y is the described variable, x is the explanatory variable and the distribution expresses the nature of the uncertainty. □

Example *4: If even the distance is under uncertainty (e.g. the way is uncertain due to road closures), the model will be*

$$f(x, y) = f(x) f(y|x) \qquad (1)$$

where $f(x)$ expresses the probabilities of individual distances and $f(y|x)$ models the travel time on condition that x is as it is. □

In a standard situations we often want to explain the behavior of a multidimensional variable $y = [y_1, y_2, \cdots y_n]$ by means of more explanatory variables $x = [x_1, x_2, \cdots, x_m]$. Construction of such model is more demanding, in many cases even impossible. However, if we can assume the variables independent, we can make do with just scalar models. The formula is

$$f(y|x) \propto \prod_{i=1}^{n} f(y_i) \prod_{j=1}^{m} \frac{f(y_i|x_j)}{f(y_i)}$$

Proof

$$f(y_1 \cdots y_n | x_1 \cdots x_n) = \prod_i f(y_i | x_1 \cdots x_n) \propto \prod_i f(x_1 \cdots x_n | y_i) f(y_i) =$$

$$= \prod_i f(y_i) \prod_j f(x_j|y_i) \propto \prod_i f(y_i) \prod_j \frac{f(y_i|x_j)}{f(y_i)}$$

1.4 Model for classification

In the above models, the explanatory variable x is known and according to its values the target y is modeled. If it is not known, it must be estimated. And it is the case of classification. Here, the explanatory variable is denoted by c and it is called a pointer (because it points at the class to which y is to be classified). In this case, both y and c are unknown, so the model (corresponding to (1)) is

$$f(y, c) \propto f(y|c) f(c) \tag{2}$$

where $f(c)$ expresses our uncertainty about the classes and $f(y|c)$ says to which class y belongs.

Remark: If the classes are equally probable, then $f(x)$ is uniform and we need only the distribution $f(y|c)$.

Lecture 2

1.5 Classification

In classification we ask: Given the measured value of y, what is the class to which it belongs, i.e. what we need is the probability function

$$f(c|y)$$

and how can we construct it using the model (2)?

It holds

$$f(c|y) \propto f(y, c) \propto f(y|c) f(c)$$

Example *1: Let us have $y \in \{1, 2, 3\}$ with two equally probable classes with the model*

$f(y\|c)$	$c=1$	$c=2$
$y=1$	*0.2*	*0.4*
$y=2$	*0.7*	*0.3*
$y=3$	*0.1*	*0.3*

Perform classification of the dataset $y=\{1,3,2,3,2\}$.

The classes are given by the maximum probability in the rows:

$$y=1\rightarrow c=2$$

$$y=2\rightarrow c=1$$

$$y=3\rightarrow c=2$$

For the dataset $y=\{1,3,2,3,2\}$, *the classification is* $c=\{2,2,1,2,1\}$. □

Example *2: Let us have multimodal system with two modes described by the models (components)* $f_1(y)$ *and* $f_2(y)$ *we denote* $f(y)$[1]

$$f_1(y)=N_y(\mu=5,r=1)$$

$$f_2(y)=N_y(\mu=2,r=1)$$

The probability of the first class is $f(c=1)=0.2$ *and the second is* $f(c=2)=0.8$. *Perform classification of the dataset* $y=\{4.3,\ 2.1,\ 3.4\}$

For the classification we maximize

$$f(c|y)\propto f(c,y)=f(y|c)f(c)$$

with respect to c.

From this we have

y	$f(y\|c=1)f(c=1)$	$f(y\|c=2)f(c=2)$	*class*
4.3	$\underline{0.062}$	*0.022*	*1*
2.1	*0.001*	$\underline{0.317}$	*2*
3.4	*0.022*	$\underline{0.119}$	*2*

and we select the components with the grater value in the rows. □

[1] *It is*

$$N_y(\mu,r)=\frac{1}{\sqrt{2\pi r}}\exp\left\{-\frac{1}{2r}(y-\mu)^2\right\}$$

Example 3: *Why naive Bayes!*

We have 3-dimensional multinomial variable $y = [y_1, y_2, y_3]$ with categorical distribution (i.e. each triple $[y_1, y_2, y_3]$ has its probability $p_{1,2,3}$) with $y_1 \in \{1, 2, \cdots, 5\}$, $y_2 \in \{1, 2, \cdots 8\}$ and $y_3 \in \{1, 2, \cdots 6\}$. The model for a single component is given by a vector of probabilities for each combination of the values of y, i.e.

y_1	y_2	y_3	$p_{1,2,3}$
1	*1*	*1*	·
1	*1*	*2*	·
	...		
1	*2*	*1*	·
	etc.		

This table has 5·8·6 = 240 rows (only for one component).

If we assume independency of y, the description is given by three vectors with total dimension 5+8+6 = 19, which is substantially less then before. And the joint probability is given by the product of marginals. □

That is, why Naive Bayes method is so useful. And moreover, its quality is surprisingly good even if the condition of independency is not absolutely true.

Lecture 3

1.6 Naive Bayes classification

We assume that variables in y are independent. Then we have

$$f(c|y) \propto f(c, y) = f(c) f(y|c) = f(c) \prod_{i=1}^{n} f(y_i|c) \tag{3}$$

and only $f(c)$ and scalar models of individual variables $f(y_i|c)$ are necessary. For equal probabilities of the classes, even $f(c)$ disappears (into the $\propto$ sign).

In the previous examples we assumed that the models of the components $f_j(y|c)$ as well as the probabilities of the classes $f(c)$ are known. In reality, they are not and have to be estimated. A standard Naive Bayes method performs the estimation with the teacher, i.e. with learning data. If the learning data are not at disposal - the learning without the teacher must be used. It is based on mixture estimation.

Example

Let us have the following learning dataset D_L of the measured values of the variable $y_t = [y_1, y_2]_t$ supplied by the values of the pointer c_t indicating to which the data record belongs; it is $c_t \in \{1, 2\}$ - two components

$$D_L = \{y_t, c_t\}_{t=1}^{nL} = \{y_{1;t}, y_{2;t}, c_t\}_{t=1}^{nL}$$

e.g

t	y_1	y_2	c
1	3.4	8.1	1
2	3.2	8.5	1
3	9.7	1.3	2
4	2.9	8.6	1
5	9.5	1.1	2

We define models

$$f_1(y_{1;t}|c_t = 1),\ f_1(y_{1;t}|c_t = 2),\ f_1(y_{2;t}|c_t = 1) \ \ \textit{and}\ \ f_2(y_{2;t}|c_t = 2)$$

for the first and second variable y and the first and second component. As we know the values of the pointer.

Now, the data $C^{variable}_{component}$ belonging to the first model are $C^1_1 = \{3.4, 3.2, 2.9\}$ and similarly for the second model (first variable and second component) $C^1_2 = \{9.7, 9.5\}$ and $C^2_1 = \{8.1, 8.5, 8.6\}$ and finally $C^2_2 = \{1.3, 1.1\}$. These data can be used for estimation pf the individual models.

For normal constant components $y_{i;t} = \mu^i_j + e_{i;t}$ (i - variable, j - component), the estimates are

$$\mu^1_1 = \frac{3.4 + 3.2 + 2.9}{3} = 3.166,\ \ \mu^1_2 = \frac{9.7 + 9.5}{2} = 9.6$$

$$\mu^2_1 = \frac{8.1 + 8.5 + 8.6}{3} = 8.4,\ \ \mu^2_2 = \frac{1.3 + 1.1}{2} = 1.2$$

So, using the independence of y_1 and y_2, and common variance equal to 1 we have

$$f(y|c = 1) = N_{y_1}\left(\mu^1_1, 1\right) N_{y_2}\left(\mu^2_1, 1\right) =$$

$$f(y|c = 2) = N_{y_1}\left(\mu^1_2, 1\right) N_{y_2}\left(\mu^2_2, 1\right)$$

For the pointer we have

$$f(c = 1) = \frac{3}{5}\ \ \textit{and}\ \ f(c = 2) = \frac{2}{5}.$$

Then, according to (3), we have

$$f(c|y) \propto [f(c = 1) f(y_1|c = 1) f(y_2|c = 1),\ f(c = 2) f(y_1|c = 2) f(y_2|c = 2)]$$

Now, to classify e.g. the data record $y_t = [3.1,\ 8.3]$ *we have*

$$f\left(c|y_t\right) \propto \left[0.068,\ 1.4\cdot 10^{-18}\right].$$

As the first number is greater, we classify y_i*into the first class.* □

Program for more complex situation is in the file `class_4.sce`.

Naive Bayes with kernel estimation

The estimated probability density function is approximately expressed as an average of kernel functions

$$f\left(y|c\right) \doteq \hat{k} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}K\left(\frac{y-y_i}{h}\right).$$

The most frequent kernel function is the standard Gaussian density proportional to $\frac{1}{\sqrt{r}}\exp\left(-\frac{1}{2}e^2\right)$ where $e = \frac{y-\mu}{\sqrt{r}}$. Here, $\sqrt{r} = h$ is standard deviation and $\mu = y_i$ is expectation. In this way, the approximation reads

$$\hat{k} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\exp\left\{-\frac{1}{2}\left(\frac{y-y_i}{h}\right)^2\right\}$$

where y is the argument of the function, $y_i, i = 1, 2, \cdots, n$ are learning data and h sets the width of the kernel function. Its value can be chosen as[2]

$$h = \frac{\sqrt{\hat{r}}}{N^{\frac{1}{5}}}$$

where $\hat{r}$ is the sample variance and N number of the learning dataset.

Remark: The method is very simple, however, each evaluation needs using all learning data.

Example: Perform classification of multimodal data y *with three components based on kernel approximation of the component models with the learning data sorted to the clusters* $C_j, j = 1, 2, 3$.

$$C_1 = \{2.6,\ 3.1,\ 0.4,\ 2.9\}$$

$$C_2 = \{13.2,\ 11.3,\ 15.4,\ 12.8\}$$

$$C_3 = \{7.5,\ 8.2, 6.7,\ 9.1\}$$

First we can see, that the estimate of the probabilities of components are uniform (why?). The

[2]It so called rule-of-thumb.

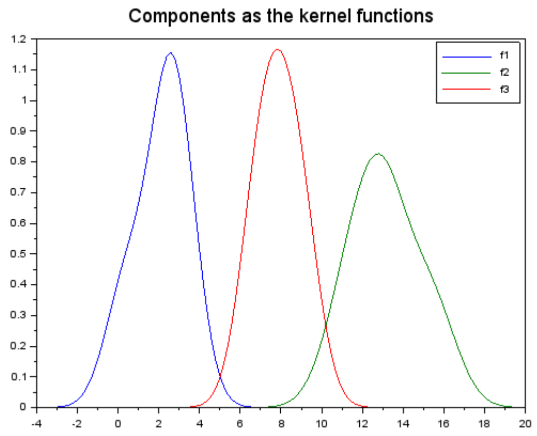
density functions of the components will be kernel approximated:

$$\hat{k}_1(y) = \frac{1}{h}\exp\left\{-\frac{1}{2}\left(\frac{y-2.6}{h}\right)^2\right\} + \frac{1}{h}\exp\left\{-\frac{1}{2}\left(\frac{y-3.1}{h}\right)^2\right\} +$$

$$+\frac{1}{h}\exp\left\{-\frac{1}{2}\left(\frac{y-0.4}{h}\right)^2\right\} + \frac{1}{h}\exp\left\{-\frac{1}{2}\left(\frac{y-2.9}{h}\right)^2\right\}$$

where $h = 1.03$ for all kernels.

Similarly $\hat{k}_2(y)$ and $\hat{k}_3(y)$. We get



Remark: Notice, that the kernel functions are not Gaussian functions - they approximate the distribution of the points.

Now, the classification is very natural

$$f(c|y) \propto f(y|c), \ c = 1, 2, 3$$

We insert the value of the classified variable into each component and classify to that with the greatest value.

E.g. for $y = 6.4$ we have

$$f(c|6.4) = [0.0024, \ 0.0001, \ 0.6919]$$

and we classify into th third class. $\square$

Remark: The averages of data in the learning set are $[2.1, \ 13.75, \ 7.875]$. It can be seen that the value $y = 6.4$ is closest to the third component.

If $y = [y_1, \ y_2, \cdots, y_n]$ is multivariate then we perform the above procedure for each variable y_i

and in the end, we perform the product

$$f\left(y|c\right)=\prod_{i=1}^{n}f\left(y_i|c\right),\,\forall c$$

Again, we classify to the class with the maximum probability.

Another example is solved in the file

Naive Bayes with parametric estimation

The kernel estimation is elegant but very demanding. Instead of kernels we will describe the components by some distribution and will estimate its parameters. We will proceed with normal distributions. We will demonstrate it with the same example as before but with two variables y.

Example (as before): We have the learning data in clusters $C_{component}^{variable}$

$$C_1^1=\{2.6,\,3.1,\,0.4,\,2.9\}\,;\,C_2^1=\{13.2,\,11.3,\,15.4,\,12.8\}\,;\,C_3^1=\{7.5,\,8.2,6.7,\,9.1\}$$

for the first variable y_1 *and*

$$C_1^2=\{12.5,\,13.9,\,11.4,\,12.7\}\,;\,C_2^2=\{3.2,\,1.3,\,5.4,\,2.8\}\,;\,C_3^2=\{9.5,\,7.2,8.7,\,6.1\}$$

for the second one. Perform classification of the value $y=[10.8,\,6.3]$ *using normal components with Naive Bayes estimation.*

First we estimate the components of individual variables. Gaussians are given by expectation and variance. They can be estimated from the learning data

$$m^1=[2.25,\,13.175,\,7.875]\,,\;\;m^2=[12.625,\,3.175,\,7.875]$$
$$r^1=[1.56,\,2.87,\,1.04]\,,\;\;r^2=[1.05,\,2.87,\,2.31]$$

with $f\left(c\right)$ *being uniform.*

Inserting the tested point $y=[10.8,\,6.3]$ *into the component models* $N_{y_1}\left(m^i\left(j\right),r^i\left(j\right)\right)$ *-* i *is variable and* j *is component, we get*

$$f\left(c|y_1\right)\propto\left[2.2\cdot10^{-14},\,8.8\cdot10^{-2},\,6.4\cdot10^{-3}\right]$$

$$f\left(c|y_2\right)\propto\left[1.7\cdot10^{-3},\,6.2\cdot10^{-5},\,1.2\cdot10^{-1}\right]$$

and for uniform $f\left(c\right)$ *the distribution* $f\left(c|y\right)\propto f\left(y_1|c\right)\left(y_{2|c}\right)$

$$f\left(c|y\right) \propto \left[3.8 \cdot 10^{-14},\, 5.5 \cdot 10^{-6},\, 7.7 \cdot 10^{-4}\right]$$

From the last distribution we see that the classification for y is the third class. $\square$

Remark: It is also possible to perform the estimation of the component parameters on-line. Then, for each component j, define zero initial statistics S_j (sum) and κ_j (count). Then with coming data y_t from the $\hat{j}$-th class, perform update of the $\hat{j}$-the statistics

$$S_{\hat{j};t} = S_{\hat{j};t-1} + y_t, \;\; \kappa_{\hat{j};t} = \kappa_{\hat{j};t-1} + 1,$$

other statistics for $j \neq \hat{j}$ do not change.

The estimate of expectations is $\hat{m}_j = S_{j;N_j}/\kappa_{j;N_j}$, where N_j is the number of data in the j-the component. Variances can be set small and fixed.

Lecture 4

1.7 Mixture estimation

If we do not have a learning dataset (learning without teacher), we can use mixture estimation. Under the Naive Bayes methodology, we can deal only with one classified variable. For multivariate one, the product of individual distributions is the solution.

The basic problem with mixture estimation without the knowledge of active components is that we do not know which component is to be updated with the newly measured y. That is why, after measuring y, we need first to make probabilistic classification (to determine probabilities that the y belongs to individual components) and then to update the components with the ratio of y corresponding to the component probability.

That is, for the component

$$f_j\left(y|c=j\right)$$

and the measured $y = y_t$ we compute weights

$$w_j = f\left(c=j|y_t\right) \propto f\left(y_t|c=j\right) f\left(y_t\right),\; j = 1, 2, \cdots, n_c$$

Then we can update the statistics (for constant normal components with statistics S (sum) and κ (count))

$$S_{j;t} = S_{j;t-1} + w_j y_t$$

$$\kappa_{j;t} = \kappa_{j;t-1} + w_j$$

for all components j.

Finally, the estimates are

$$\hat{m}_{j;t} = S_{j;t}/\kappa_{j;t},\;\; \forall j.$$

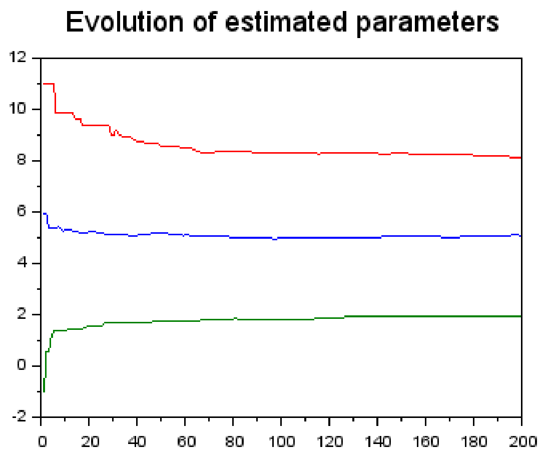
The classification is given as argument minima weights w

$$c_p = \arg\min w$$

Example program

```
// Scalar mixture estimation
// - aL is not estimated
// - better weights under logarithm
// ------------------------------------
exec('SCIHOME/ScIntro.sce',-1); mode(0);

nd=200;

// Simulation
aLS=[.4 .3 .3];                         // switching parameters
mS=[5 2 8];                             // comp. expectations
sdS=[.6 .3 .8];                         // comp. variances
nc=length(aLS);                         // number of components
for t=1:nd                              // loop for simulaion
  c(t)=sum(cumsum(aLS)<rand(1,1,'u'))+1;     // pointer
  y(t)=mS(c(t))+sdS(c(t))*rand(1,1,'n');     // output
end

//Estimation
m=[7 -1 11];                            // prior expectations
ka=[1 1 1];                             // counter
S=m.*ka;                                // sum
for t=1:nd                              // loop for estimation
  for j=1:nc
    q(j)=GaussN(y(t),m(j),.1);          // proximities
  end
  w=q/sum(q);                           // weihts
  cp(t)=amax(w);
  wt(:,t)=w;
  for j=1:nc
    S(j)=S(j)+w(j)*y(t);                // statistics
    ka(j)=ka(j)+w(j);                   //   update
    m(j)=S(j)/ka(j);                    // estimates
  end
  mt(:,t)=m;
end

// Results
mS,m                                    // final estimates
Acc=acc(c,cp)                           // accuracy of classif.
set(scf(),'position',[200 300 400 300]) // evol. of parameters
plot(mt')
title('Evolution of estimated parameters','fontsize',4)
```

Evolution of estimated expectations



Lecture 5

2 Regression

Here, we will demonstrate the logistic and Poisson regression. They are both very similar:

1. They use nonlinear models with unknown parameters.
2. Their estimation is performed off-line using numerical optimization. It has two phases: learning and testing.
3. They need to cope with non-negativity of estimated parameters.

2.1 Logistic regression

Model for variable c_t with Bernoulli distribution

$$f(c_t|p) = p^{c_t}(1-p)^{1-c_t}$$

with $c_t = 0, 1$ is dichotomous model output $p \in (0,1)$ is the probabilistic parameter: $p = P(c_t = 1)$.

The expectation of c_t is

$$E[c_t|p] = p$$

Now, we would like to extend this model so that its expectation will be modeled by regression in the form

$$p \to x'b = b_0 + b_1 x_1 + \cdots + b_m x_m$$

However, there are problems. $p \in (0,1)$, i.e. it is nonnegative and bounded from above.

1. The solution with respect to bounding is: instead of p to model $\frac{p}{1-p}$ which is from the interval $(0, \infty)$
2. Nonnegativity of $\frac{p}{1-p}$ can be solved by taking logarithm $\ln \frac{p}{1-p}$. This variable is called *logit*
$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

This logit will be modeled by regression

$$\ln\left(\frac{p}{1-p}\right) = x_t b$$

The final model $f(c_t|b)$ can be derived from the above expression and it has the form

$$f(c_t|b) = p = \begin{cases} \frac{\exp\{x_t b\}}{1+\exp\{x_t b\}} & \text{for } c_t = 1 \\ \frac{1}{1+\exp\{x_t b\}} & \text{for } c_t = 0 \end{cases}$$

and using the fact that $c_t \in \{0,1\}$ we can write the model as

$$f(c_t|b) = \frac{\exp\{c_t x_t b\}}{1+\exp\{x_t b\}}.$$

Note, that both the mentioned demands are fulfilled - $p \in (0,1)$, and nonnegative, indeed.

For estimation of the parameter p we will construct the likelihood function

$$L_N(p) = \prod_{t=1}^{N} \frac{\exp\{c_t x_t b\}}{1+\exp\{x_t b\}}$$

where we used a trick for writing the model in a unified form. For $c_t = 1$ the nominator in the model will be $\exp\{x_t b\}$ and for $c_t = 0$ it will be 1.

The log-likelihood is

$$\ln L_N(p) = \sum_{t=1}^{N} \left[c_t x_t b - \ln\left(1+\exp\{x_t b\}\right)\right]$$

As the first and second derivatives of this expression can be computed analytically, the Newton method for numerical maximization is very suitable. It is quick and has fast convergence.

Program for experimentation in Scilab is `DM_LogisReg.sce`

2.2 Poisson regression

Model with Poisson distribution

$$f(c_t|\lambda) = \exp\{-\lambda\} \frac{\lambda^{c_t}}{c_t!} \tag{4}$$

with $c_t = 0, 1, 2, \cdots, \infty$, $\lambda > 0$ it the expectation (average number of events per time unit). Again, the expectation should be expanded by regression. The condition of upper limit is nor demanded, but the non-negativity remains and is solved in the same way as for logistic regression - by expanding logarithm of λ instead of λ itself

$$\ln(\lambda) = x_t b = b_0 + b_1 x_1 + \cdots + b_m x_m.$$

Thus, for λ we have

$$\lambda = \exp\{x_t b\}.$$

The final model $f(c_t|\lambda)$ will be (4) with the above substitution - for log-likelihood we express directly its logarithm

$$\ln\{f(c_t|b)\} = -\exp\{x_t b\} + c_t x_t b - \ln(c_t!)$$

Log-likelihood is

$$\ln L_N(b) = \sum_{t=1}^{N} \left[-\exp\{x_t b\} + c_t x_t b - \ln(c_t!)\right]$$

and it is maximized numerically.

Program to the Poisson regression is here

3 Classical clustering

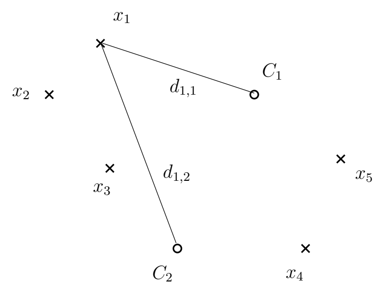
The task of clustering consists in dividing the data space into several subspaces whose data are somehow similar. Mostly the similarity is given by the distance of the points. We demand that the points in a cluster are as close as possible and on the other hand the points between different clusters are as remote as possible. However, the sorting can be governed also by other rules as e.g. color or shape of "data points".

For us the clustering according to the distance will be decisive. The distance is mainly Euclidean but it can also be some other, like Manhattan or Minkowski ones.

3.1 K-means algorithm

Let us have a data sample $X = [x_1, x_2, \cdots, x_N]$ where individual entries are data records (point) $x_t = [x_{1;t}, x_{2;t}, \cdots, x_{n;t}]$, N is total number of data records and n is the number of variables in the dataset. The algorithm of clustering is as follows

0. Determine the number of clusters ans set their initial centers.
1. Measure the distance from each data point to each cluster center and assign the point to the nearest center. The points form clusters.
2. Compute the average of points in each cluster and set it as its new center.
3. Check, if the centers changed. If yes, go to 1. If not, the algorithm ends.



x - points, C - centers
$d_{1,1} < d_{1,2} \rightarrow x_1$ belongs to C_1

Program

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

```
x=[1.2 2.5 6.5 7.8 9.3];
c=[5 7];
cOld=c;
printf('d = distance of points from clusters\n')
printf('C = occupancy of clusters\n\n')

for ite=1:10
  printf('Iteration ---------- %d\n',ite)
  for i=1:2
    for j=1:5
      d(i,j)=abs(c(i)-x(j));
    end
  end
  d
  C=list(); C(1)=[]; C(2)=[];
  for j=1:5
    [xx,k]=min(d(:,j));
    C(k)=[C(k) x(j)];
  end
  printf('\n   (centers: %5.2f  %5.2f)\n',c)
  C

  c(1)=mean(C(1));
  c(2)=mean(C(2));

  if sum(abs(cOld-c))<.01
    break
  end
  cOld=c;
end
```

Description of the program

Definition of the distance

Simulation

Three centers m, standard deviation of data in clusters sd are set. Two dimensional data generated in loop. In the i-th cluster $n(i)$ data points are simulated.

Algorithm

Structure variable C is defined. It has items $.c0$ - initial centers, $.c$ - new centers, $.cs$ - centers from previous step, $.cd$ - points in a cluster. It runs according to the list above.

3.2 K-medoids algorithm

This algorithm is similar to k-means with the difference, that centers (medoids) are always data points. The algorithm is:

0. Determine md as the desired number of clusters. Randomly select md data points as initial centers of medoids. The rest are non-medoids.
0. For all medoids and all non-medoids determine distances of points (non-medoids) and medoids. Overall distance is the sum of all distances.
1. For all medoids and all non-medoids perform:
   (a) Experimentally swap medoid and non-medoid and determine overall distance.
   (b) For the attempt with minimal overall distance, definitely swap the medoid and non-medoid and continue in the algoritm.
2. If the overall distance after swapping is smaller than the previous one, continue by 1. If not, iterations end.
3. To each medoid find the points (non-medoids) that are closest to it. They will form the final clusters corresponding to individual medoids.

3.3 Fuzzy clustering

C-means algorithm

In the c-means algorithm we minimize criterion

$$J = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} \|x_i - c_j\|^2, \ m \geq 1$$

where u_{ij} is a degree of membership of the point x_i to cluster c_j and $\| \cdot \|$ is a norm.

The update of weights u_{ij} is performed as follows

- determine the centers (follows from minimization of the criterion)

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{m} x_i}{\sum_{i=1}^{N} u_{ij}^{m}}$$

- weights (are given as membership functions)

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \tag{5}$$

Remarks

1. The formula for c_j follows from minimization the criterion J.
2. u_{ij} says hoe strongly the point x_i belongs to the center c_j
3. $\frac{\|x_i - c_j\|}{\|x_i - c_k\|}$ is the distance of x_i from c_j relative to the distance of x_i from some other center c_k.
4. $\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}$ is the share of the x_i to c_j distance in total distance of x_i from all centers.

Algorithm

0. Set the initial matrix of membership U.
1. Compute the centers c_j with existing matrix U.
2. Update the matrix U.
3. If $\|U_{nová} - U_{stará}\| < \epsilon$, END otherwise go to 1.

Program

```
// 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

u=[2 2 2 1 1
   1 1 1 2 2];                          // initial weights

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

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

```
// recomputation of weights
  uOld=u;
  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

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

Program description

Function definitions

- CMupdt computes distances of points from centers. First normalizes over clusters and then over points. Finally creates clusters using the weights *un*.

- clusters constructs clusters according to the distances *dm*.

Simulation - standard

Initialization - updating of clusters (new centers)

Iterations - update of clusters (new clusters). Check for end of the algorithm.

Lecture 7

3.4 Density based clustering

Dbscan

We have a set of data $X = \{x_1, x_2, \cdots, x_N\}$, where $x_i \in R^m$

We define:

- **Distance** of two points x and y and denote it by $d(x,y)$.
- ϵ-**neighborhood** of point x
$$O_\epsilon(x) = \{x \in X : d(x,y) < \epsilon\}.$$
- **Inner point** is such one that has in its neighborhood at least given number of points.
- A point y is **accessible** from the point x, if a sequence of inner points from x to y exists.
- A **connection** between points x a y exists, it both these points are accessible from some inner point.

Algorithm of clustering

1. For each point from X find its ϵ-neighborhood.
2. Define variables "clus" (clusters) and "buff" (buffer) for storing points.
3. To "clus" put a single inner point and to "buff" its neighborhood.
4. Select one point (e.g. the first one) from "buff". Add it to "clus" and its neighborhood add to "buff".
5. From "buff" remove all points that already have been used (those that are in some cluster).
6. Repeat from 4. until "buff" is empty. Otherwise continue.
7. Remember the created cluster "clus" and empty the variable "clus" for continuation of the algorithm.
8. If there exists another free inner point, put it to "clus" and go to 4. If not, stop the algorithm.

Clusters are those remembered from "clus".

Example 1

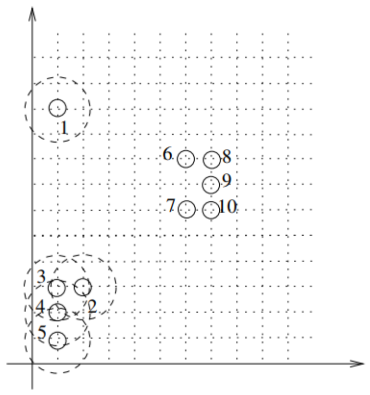
$X = \{1,2,3,5,6\}$ $\epsilon = 1.1$.

clus	*1*	*1,2*	*1,2,3*	*5*	*5,6*
buff	*2,3*	*~~1~~,3*		*6*	

Clusters are $\{1,2,3\}$ *and* $\{5,6\}$. $\square$

Example 2

Let us have 10 points as demonstrated in the picture



Points are circles and are plotted in a net with unit step. Parameter eps = 1.1, *minimum number of points is mp* = 2. *Then points*

- *3, 4, 8, 9, 10 are inner points*
- *2, 5, 6, 7 are border points*
- *1 is noise points.*

Cluster construction

If the points are two-dimensional, the best way is to draw them in a plane (as in the picture above) and to select the clusters manually. Start with arbitrary free inner point and add to it all connected points. Repeat until all points are classified.

Here the result is:

Cluster1 = {2, 3, 4, 5} a Cluster2 = {6, 7, 8, 9, 10}.

The point 1 is noise.

Program for experimentation in Scilab is `DM_dbscan.sce`

Lecture 8

3.5 Hierarchical clustering

Agglomerative clustering

There is a lot of variations of this method. We will show here one of them which is very simple. The algorithm is here:

1. All data points are denoted as clusters on the level 1 (with only one point).

2. Find two nearest clusters and join them together in one cluster. The level of the cluster is equal to the number of its points.
3. The position of the new cluster lies between the covered clusters in the ratio of their levels. The height is equal to the total distance of the covered points.
4. Remember the points (clusters) from which the new one has been created .
5. Repeat from 2 until only one cluster remains.

Example

Let us have points

$$X = \{2.5, 0.9, 2.9, 1.2, 3.8\}$$

i	1	2	3	4	5
x_i	2.5	0.9	2.9	1.2	3.8

Perform hierarchical (agglomerative) clustering.

Step 1: Distances

pts	2	3	4	5
1	1.6	0.4	1.3	1.3
2	–	2	0.3	2.9
3	–	–	1.7	0.9
4	–	–	–	2.6

Minimum is 2-4: 0.3; $\rightarrow$ *new point 6*$=\{2,4\}$ *weight* $=$ *2; position* $= \frac{0.9+1.2}{2} = 1.05$

We have

i	1	x	3	x	5	6
x_i	2.5	x	2.9	x	3.8	1.05

Step 2: Distances

pts	3	5	6
1	0.4	1.3	1.45
3	–	0.9	1.85
5	–	–	2.75

Minimum is 1-3: 0.4; $\rightarrow$ *new point 7*$=\{1,3\}$ *weight* $=$ *2; position* $= \frac{2.5+2.9}{2} = 2.7$

We have

i	x	x	3	x	5	6	7
x_i	x	x	x	x	3.8	1.05	2.7

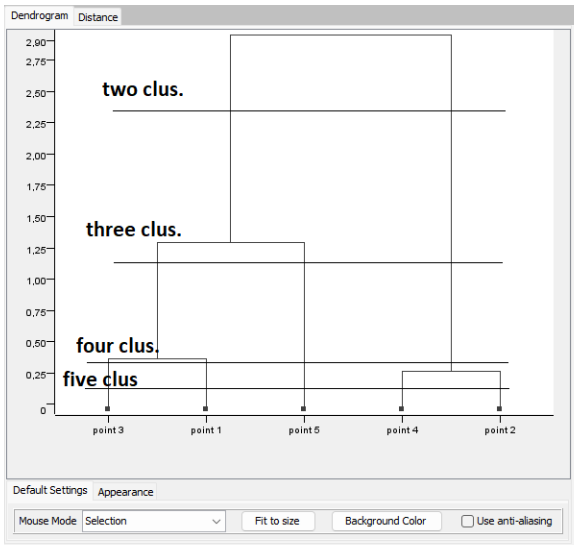
Step 3: Distances

pts	*6*	*7*
5	*2.75*	*1.1*
6	*–*	*1.65*

Minimum is 5-7: 1.1; $\rightarrow$ *new point 8=*$\{5,7\}$ *weight = 1+2=3; position =* $\frac{3.8+2\cdot 2.7}{3} = 3.067$

Step 4: The last one: new point 9=$\{6,8\}$ *weight = 2+3 = 5; position =* $\frac{2\cdot 1.05+5\cdot 3.067}{7} = 2.491$

The ***dendrogram*** *is in the following picture*



Remark: The height of the forks is equal to the total distance of th points in the cluster (i.e. dissimilarity in the cluster). But the vertical distances are not so much important. What is important is the hierarchy.

Program for experimentation in Scilab: `DM_hierAgl.sce`

Divisive clustering

In divisive clustering we proceed from top to bottom. We start with one cluster that contains all data points and subsequently divide clusters so that there would be minimal point distances in clusters and maximal distances between clusters. For a given definition of the distance $D(x,y)$ we introduce following notions

Big cluster C_T - is a cluster to be divided.

Left and right cluster C_L a C_R - clusters created by division

Distance between clusters C_L and C_R denoted by I_{LR}

Distance inside clusters - U_L, $U_{\acute{R}}$

Distance of the divided cluster - $U_T = I_{LR} + U_L + U_R$ (it is sum of distances from each point from C_L to each point from C_R - it is independent on division)

Task: Find C_L a C_R so that

$$H_{LR} = (1-\alpha)\underbrace{I_{LR}}_{H_1} - \alpha\underbrace{[U_L + U_R]}_{H_2} \to \min$$

This task is combinatorial and it is np-hard. For its approximative numerical solution we will use the method called

Avalanche method.

We have a cluster C_T (in the beginning the whole data sample), which is to be divided.

We introduce C_L as an empty set and C_R as the whole cluster C_T.

1. In C_R we find anti-medoid - i.e. the point which is maximally remote from all other points in the cluster C_R.
2. Shift anti-medoid into the cluster C_L and compute the value of the criterion H_{LR}.
3. Try to add another point that is closest to the previously added one.
4. If the value of the criterion increases we leave the point in C_L and we go to the point 3. If it dos not increase, the algorithm ends.

Program for experimentation in Scilab: `DM_hierDiv.sce`

Lecture 9

4 Classical classification

By classification we mean assignment of a data record (point) to some cluster or more clusters each with its probability. Here, we mostly assume, that clusters have already been created by some clustering method.

4.1 K-nearest neighbour

It is a basic form of classification.

We have data $X = \{x_i\}_{i=1}^N$ with detected clusters. The task is: assign a newly measured data point y to some cluster.

The procedure of classification is the following:

1. Compute the distance of the point y from all points from $x_i \in X$.
2. Determine k points x_i, $i = 1, 2, \cdots, k$ nearest to y.
3. Assign y to the cluster to which majority of the k nearest points belongs.

Remark: If there are more than one such cluster, take the first of them.

Program for experiments in Scilab is `DM_knearest.sce`

4.2 Decision tree

Let us have discrete data records $x_t = [x_1, x_2, \cdots, x_n]_t$, $t = 1, 2, \cdots, N$ and a pointer variable $c_t \in \{1, 2, \cdots, m\}$ which is a label of the class (cluster) to which the record x_t belongs (learning with the teacher).

The principle of tree construction is as follows:

We construct a matrix from the data records and add the pointer variable c_t as its last column. We have matrix $N \times (m + 1)$

$$X = [x_{ti}, \, c_t], \, t = 1 : N, \, i = 1 : m$$

We chose some variable x_i and according to its values we sort the remaining parts of the matrix into groups. Then, in each group we again select a variable and do the same. We repeat this procedure until each group contains only the same value of the pointer. If some final group has more than one pointer value, the decision is probabilistic.

It is clear that the subsequent choice of variables is very important for a success of the task. However, the proper choice is a combinatorial task for which we need to use some heuristic methods. One of them is illustrated in the following example.

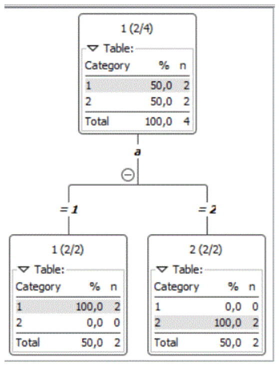
Example

Let us have the following data

t	x_1	x_2	c
1	*1*	*1*	*1*
2	*1*	*2*	*1*
3	*2*	*1*	*2*
4	*2*	*2*	*2*

where x_1, x_2 are data records and c is pointer variable.

It is evident, the variable x_1 decides about the classification (on the basis of only the variable x_1 we can decide about classes of all records). The tree for the order of variables x_1 - x_2 is



1 (2/4)

Table:

Category	%	n
1	50,0	2
2	50,0	2
Total	100,0	4

1 (2/2)

Table:

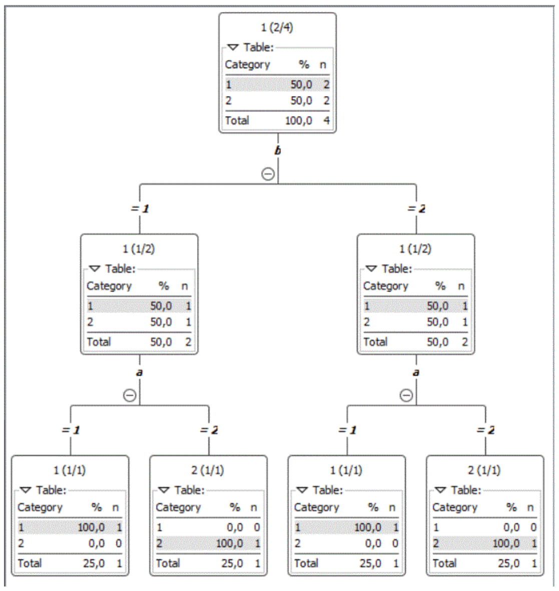
Category	%	n
1	100,0	2
2	0,0	0
Total	50,0	2

2 (2/2)

Table:

Category	%	n
1	0,0	0
2	100,0	2
Total	50,0	2

If we swap the order of variables to x_2 - x_1 we get the tree longer and more complex



1 (2/4)

Table:

Category	%	n
1	50,0	2
2	50,0	2
Total	100,0	4

1 (1/2)

Table:

Category	%	n
1	50,0	1
2	50,0	1
Total	50,0	2

1 (1/2)

Table:

Category	%	n
1	50,0	1
2	50,0	1
Total	50,0	2

1 (1/1)

Table:

Category	%	n
1	100,0	1
2	0,0	0
Total	25,0	1

2 (1/1)

Table:

Category	%	n
1	0,0	0
2	100,0	1
Total	25,0	1

1 (1/1)

Table:

Category	%	n
1	100,0	1
2	0,0	0
Total	25,0	1

2 (1/1)

Table:

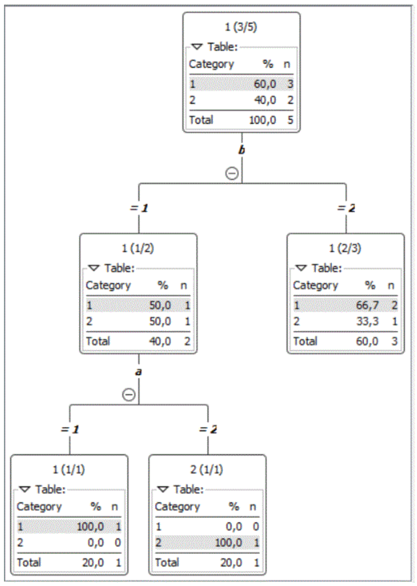
Category	%	n
1	0,0	0
2	100,0	1
Total	25,0	1

However, both the trees lead to deterministic decision making (the final percent are 100%).

If we supply the data by one more record (the last row of the table)

t	x_1	x_2	c
1	*1*	*1*	*1*
2	*1*	*2*	*1*
3	*2*	*1*	*2*
4	*2*	*2*	*2*
5	*2*	*2*	*1*

which is in contradiction with the others, the thee will be like this



1 (3/5)

Table:

Category	%	n
1	60,0	3
2	40,0	2
Total	100,0	5

1 (1/2)

Table:

Category	%	n
1	50,0	1
2	50,0	1
Total	40,0	2

1 (2/3)

Table:

Category	%	n
1	66,7	2
2	33,3	1
Total	60,0	3

1 (1/1)

Table:

Category	%	n
1	100,0	1
2	0,0	0
Total	20,0	1

2 (1/1)

Table:

Category	%	n
1	0,0	0
2	100,0	1
Total	20,0	1

In the second layer, the decision is probabilistic.. □

Implementation of the task in KNIME

We take an example from web https://tanthiamhuat.files.wordpress.com/2015/10/decision-tree-tutorial-by-kardi-teknomo.pdf

Example

The data bring information about the ways in which people go to work.

(6)

sex	*has a car?*	*fare*	*income*	***way***
M	*0*	*L*	*N*	***B***
M	*1*	*L*	*S*	***B***
Z	*1*	*L*	*S*	***V***
Z	*0*	*L*	*N*	***B***
M	*1*	*L*	*S*	***B***
M	*0*	*S*	*S*	***V***
Z	*1*	*S*	*S*	***V***
Z	*1*	*D*	*V*	***A***
M	*1*	*D*	*S*	***A***
Z	*1*	*D*	*V*	***A***

where "sex", "car", "fare" and "income" data records and "way" is a value of the pointer variable.

The values of variables are:

sex: M = man, W = woman;

car: 0 - does not have, 1 - has

fare: L - low, M = medium, H - high;

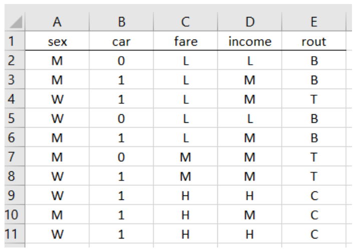
income: L = low, M = medium, H = high;

way: B - bus, T - train, C - car.

The task is to decide about the way (B, T, C) on the basis of the values in data records.

We are going to show the solution in KNIME.

1. Data can be set into table e.g. in EXCEL and exported as csv table to disk.

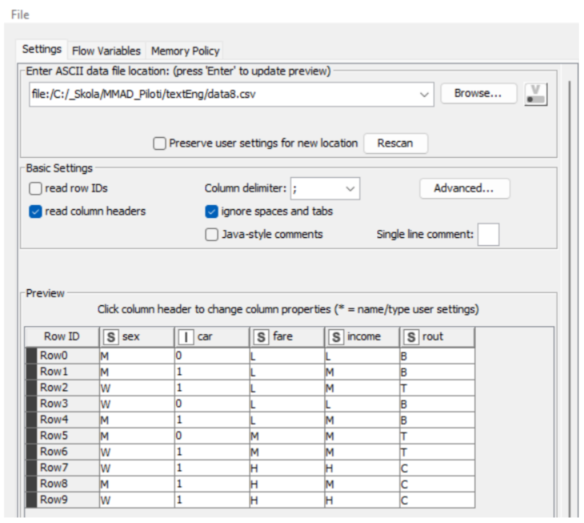


	A	B	C	D	E
1	sex	car	fare	income	rout
2	M	0	L	L	B
3	M	1	L	M	B
4	W	1	L	M	T
5	W	0	L	L	B
6	M	1	L	M	B
7	M	0	M	M	T
8	W	1	M	M	T
9	W	1	H	H	C
10	M	1	H	M	C
11	W	1	H	H	C

2. In KNIME we open a New KNIME workflow (icon new).

3. In KNIME in the left side there is a window Node Repository (here icons of various tasks are found).

   (a) In IO we find Read and File reader and drag it by mouse to the working area. An icon of the Reader appears. We click on it by left mouse button (or twice by the right) and we obtain menu Configuration



Enter ASCII data file location: (press 'Enter' to update preview)

file:/C:/_Skola/MMAD_Piloti/textEng/data8.csv

☐ Preserve user settings for new location

Basic Settings

☐ read row IDs
☑ read column headers
Column delimiter: ;
☑ ignore spaces and tabs
☐ Java-style comments
Single line comment:

Click column header to change column properties (* = name/type user settings)

Row ID	S sex	I car	S fare	S income	S rout
Row0	M	0	L	L	B
Row1	M	1	L	M	B
Row2	W	1	L	M	T
Row3	W	0	L	L	B
Row4	M	1	L	M	B
Row5	M	0	M	M	T
Row6	W	1	M	M	T
Row7	W	1	H	H	C
Row8	M	1	H	M	C
Row9	W	1	H	H	C

Here (up) we can set the name of the data csv file. Most of the rest is set automatically.
But **important** !!!

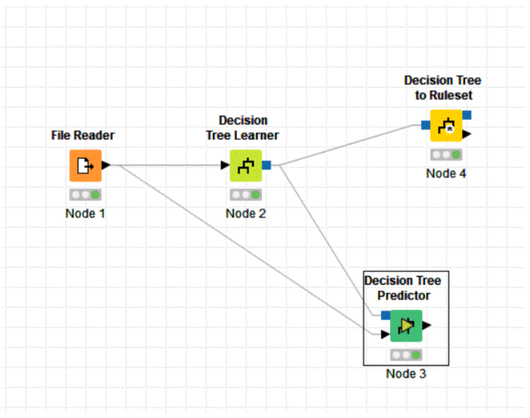
- The pointer variable must be set as a string. The rest of variables can stay as they are.
- Strings are sorted by values the other by intervals.
- The change of the variable type can be done in the menu which can be obtained by clicking at the title of the variable in the data table below. After a click a menu window appears in which the type can be selected.
- W click once again at the icon of the task and select Execute (or press F7).

(b) Next, in the window Node Repository open the folder Analytics and Mining and select the tool Decision Tree Learner, drag it to working area and by mouse connect it with the Reader (by the black small triangles).
Press F7.

(c) Further, we can choose the tool Decision Tree Prediction, and possibly Decision Tree to Ruleset. The small triangles are always connected to Reader, small blue rectangles subsequently with the new tool (they generate the model of the task).

4. The results can be stored by the tool IO/Write/CCV Writer or directly checked by clicking by the left mouse and opening
in Learner the menu Decision tree view
in Prediction the menu Classified Data
in Ruleset the menu Rules table

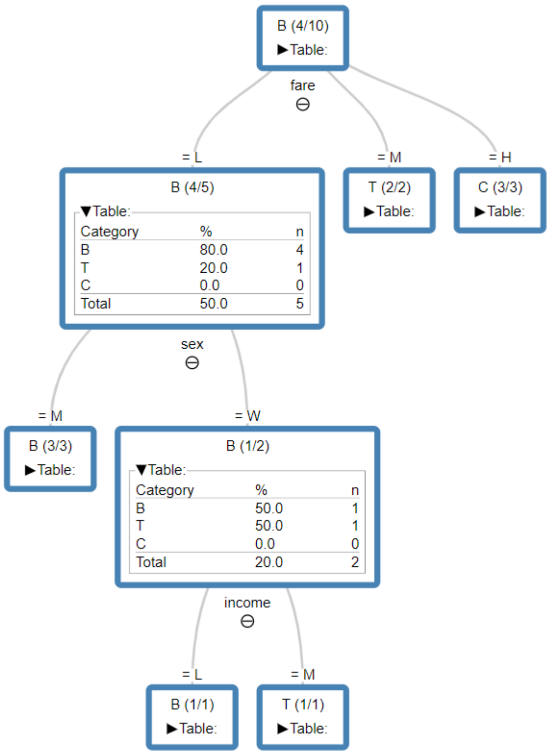
The overall view on the task in KNIME is following



Remark

If the tree ends prematurely, it is necessary to set Number of records per node = 1 in the menu Configure in the tool Decision Tree Learner. It means that the decision rule can be derived from only one data record.

The tree is here



Category	%	n
B	80.0	4
T	20.0	1
C	0.0	0
Total	50.0	5

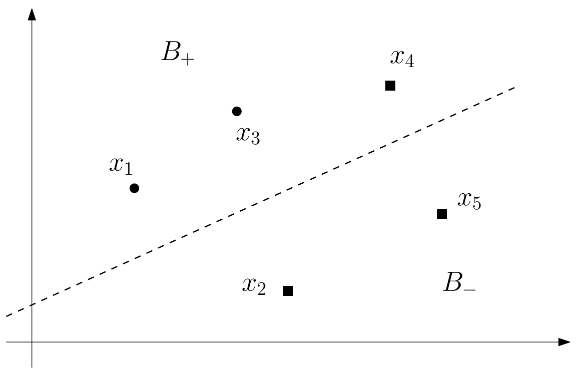
Category	%	n
B	50.0	1
T	50.0	1
C	0.0	0
Total	20.0	2

4.3 Support vector machines

In this task, we are going to find hyperplane in the data space that separates the space into two sub-spaces, one with $y = 1$ and second with $y = -1$. If the points are linearly separable, the result will be without errors. In addition, we demand so that the hyperplane would separate the points optimally. It means that the points should lay as far as possible from the hyperplane.

Theory

We will demonstrate the task in a plane (with two variables). The data sample is $X = \{x_1, x_2, \cdots, x_N\}$ where $x_i = [x_1, x_2]_i$ is i-th data record. In this case, the hyperplane will be a line as indicated in the picture



Here we have a sample of five points x_1, x_2, x_3, x_4 and x_5. The separating line is drawn dashed and it separates the points whose attributes are “circles” (up the line) and “squares” (down the line). The attributes can be expressed numerically by 1 and -1 as values of an introduced pointer variable y

x	x_1	x_2	x_3	x_4	x_5
y	1	-1	1	1	-1

The points with $y = 1$ form the set B_+, those with $y = -1$ the set B_-. So, it is

$$B_+ = \{x_1, x_3, x_4\}, \text{ and } B_- = \{x_2, x_5\}.$$

The task is to find a line which separates the points and maximizes the distance of points from itself.

Let us denote the separating line as $\alpha' x + \beta = 0$. The parallel line above it is $\alpha' x + \beta + \delta = 0$ and below it $\alpha' x + \beta - \delta = 0$ for any $\delta > 0$. All these equations are over-parameterized, i.e. can be divided by some nonzero number. We will divide them by δ and get separating line

$$w'x + b = 0$$

lines above and below

$$w'x + b \pm 1 = 0$$

For all x_1 above the above line we have the condition

$$w'x + b + 1 > 0$$

and below the below line the condition is

$$w'x + b - 1 < 0.$$

The second condition can be multiplied by -1

$$-(w'x + b) + 1 > 1$$

and using the fact that $y_i = -1$ for all x_i below and $y_i = 1$ for x_i above, we have

$$y_i (w'x_i + b) + 1 > 0$$

this single condition for all the points x_i (compare the original condition above and the modified condition below). The equality holds for parallels as borders of the above and below area.

Now, we want the above and below lines would be as far as possible one from the other. The distance of parallel lines is measured as a distance of intersections of the lines and a vertical to them. Such a vertical has equation

$$x = m + t\frac{w}{|w|}$$

where m is a fixed point, x is arbitrary point on the vertical and t is a parameter. $|w|$ is the length of w and thus $\frac{w}{|w|}$ is a unit vector. In this case the distance of the points x and m is

$$|x - m| = t\frac{|w|}{|w|} = t,$$

and it is directly equal to t. Now, we choose that x is a point on the parallel and m lies on the separating line. Then x must fulfill the equation for the parallel and m for the separating line. Tu this end we multiply the previous equation by w', add b to both sides and we obtain

$$|\underbrace{w'x + b + 1}_{=0\,\text{(parallel)}} - 1 - \underbrace{w'm + b}_{=0\,\text{(separ.)}}| = +t\frac{w'w}{|w|}$$

and the result is

$$1 = t\frac{w'w}{|w|} = t|w|$$

The distance is

$$t = \frac{1}{|w|}$$

which is to be maximized. From it the task is

$$|w| \to \min$$

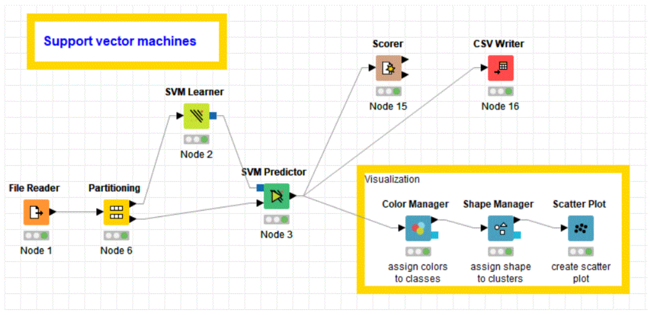
on condition that

$$y_i\left(w'x_i + b\right) + 1 > 0$$

As both w and b are to be optimized, the task is nonlinear and the solution rather complex.

Program KNIME

Is realized by the following program scheme



Block 1: Reading data.

Block 6: Division of data to learning and training parts.

Block 2: Estimation (learning).

Block 3: Prediction (classification).

Block 15: Frequencies of classification (table: from / to).

Block 16: Write results to disk.

Block in the yellow frame: Show graph of the found clusters .

Remarks

1. The results can be found in the menu which appears after clicking on the task icon.

2. The data file used can be modified directly on disk. If there are new variables (not only values), it is necessary to perform new Configuration of the data icon.

3. If the results are stored on disk, we have a possibility to investigate them in some other program - probably in Excel. To this end it is necessary to:

   (a) Set semicolon as data delimiter - in menu menu of the icon of CSV Writer, in the item Configure / Advanced.

   (b) In the menu Configure / Settings it is good to set Overwrite in the item If file exists ...

Scatter plot

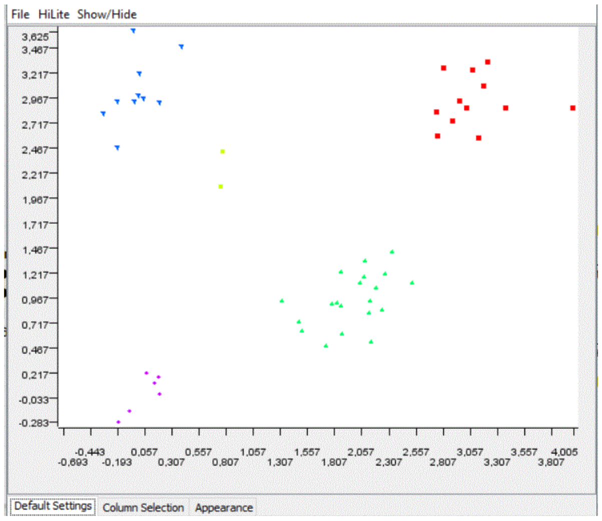
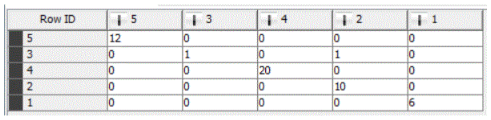


Table of classifications



Row ID	5	3	4	2	1
5	12	0	0	0	0
3	0	1	0	1	0
4	0	0	20	0	0
2	0	0	0	10	0
1	0	0	0	0	6

5 Appendix

5.1 Programs in Scilab

Elementary Scilab programs

Contents

6 Simulation and estimation

6.1 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
```

6.2 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
```

6.3 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
```

6.4 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
```

6.5 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
```

6.6 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
```

6.7 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.
```

6.8 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
```

6.9 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
```

7 Initialization

7.1 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)
```

7.2 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)
```

7.3 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)
```

8 Prediction

8.1 Zero-step prediction

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))
```

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)
```

8.2 K-step prediction

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
```

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'])
```

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
```

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
```

9 Classification

9.1 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
```

9.2 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
```

9.3 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
```

9.4 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)
```

9.5 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
```

10 Functions

10.1 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
```

10.2 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
```

10.3 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
```

10.4 Kernel function

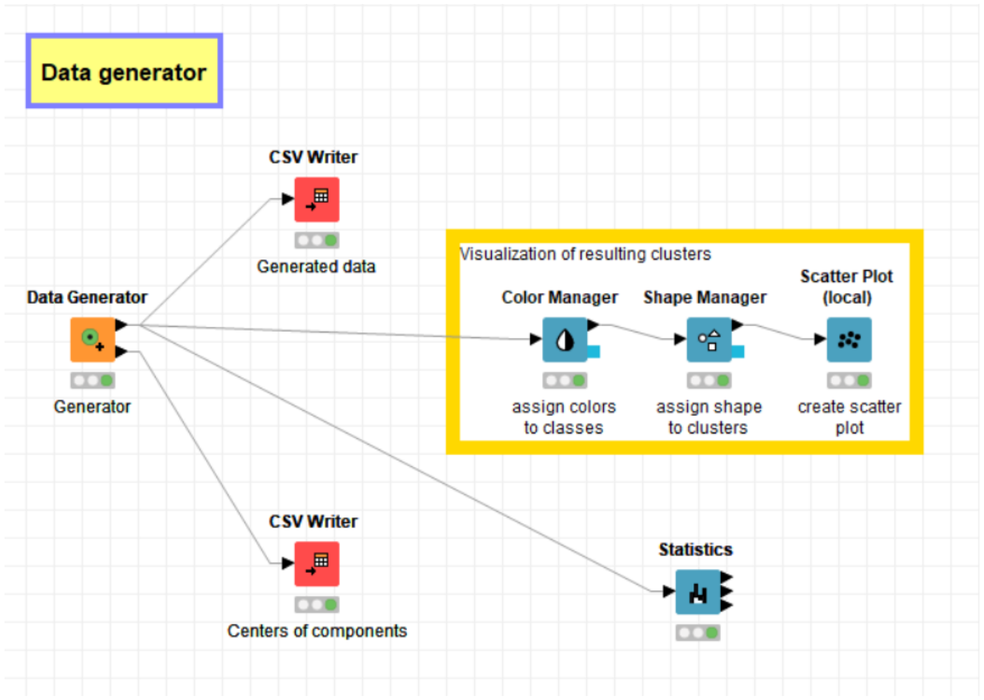
```
function [z,xx]=kerf(y,h,ik)
  // Gaussian kernel density of data y
  // r    variance
  // h    step
  if argn(2)<2, n=20; end
  if argn(2)<3, ik=0; end

  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
```

10.5 Programs in KNIME

Elementary KNIME programs

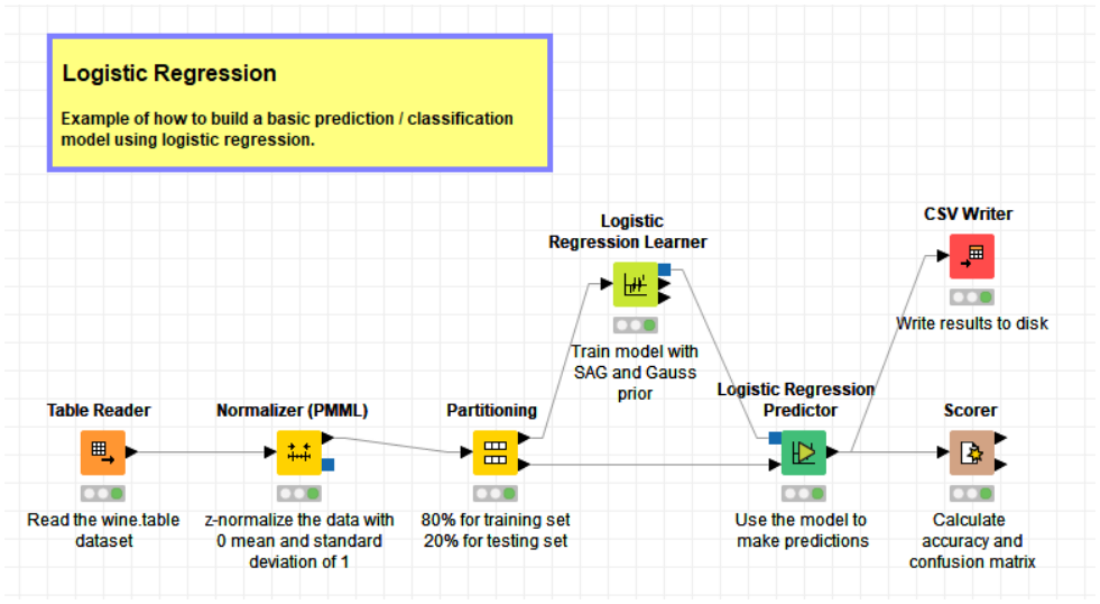
Generation of data

KNIME: Task00_dataGenerator



Logistic regression

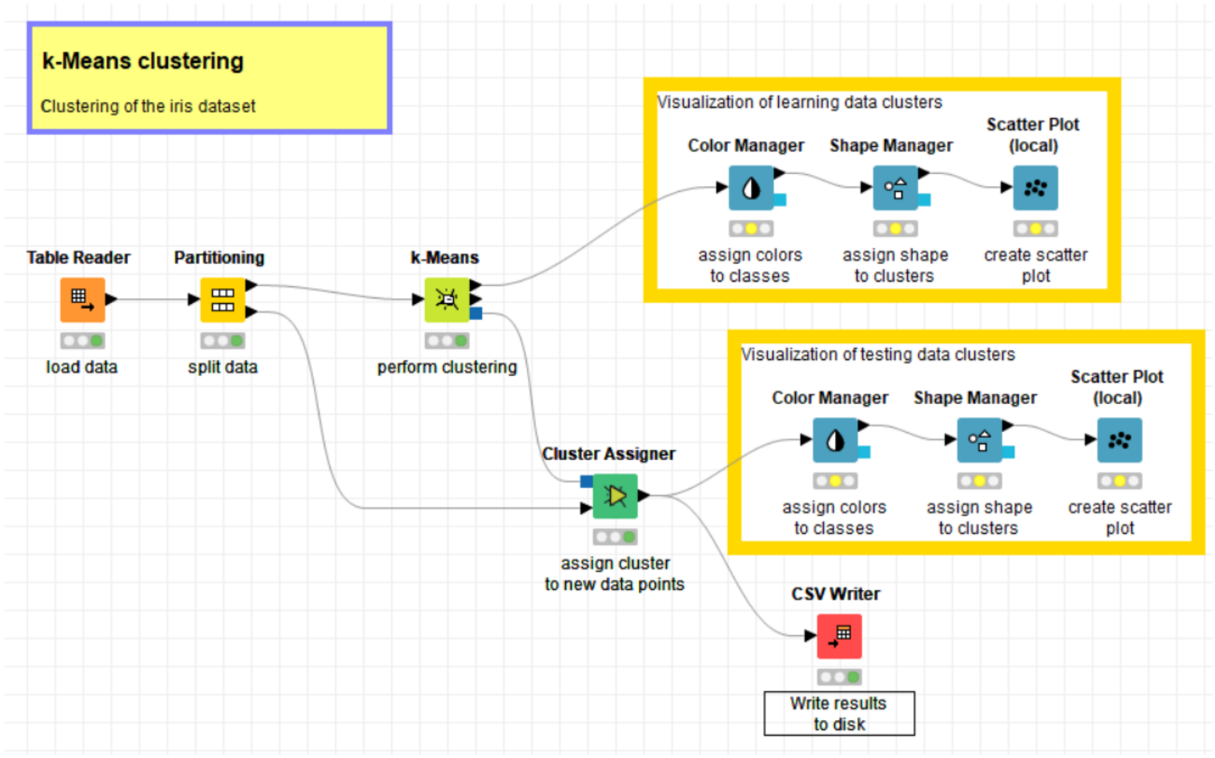
KNIME: Task01_Logistic_Regresion


Logistic Regression
Example of how to build a basic prediction / classification model using logistic regression.
Table Reader
Read the wine.table dataset
Normalizer (PMML)
z-normalize the data with 0 mean and standard deviation of 1
Partitioning
80% for training set 20% for testing set
Logistic Regression Learner
Train model with SAG and Gauss prior
Logistic Regression Predictor
Use the model to make predictions
CSV Writer
Write results to disk
Scorer
Calculate accuracy and confusion matrix

11 Clustering

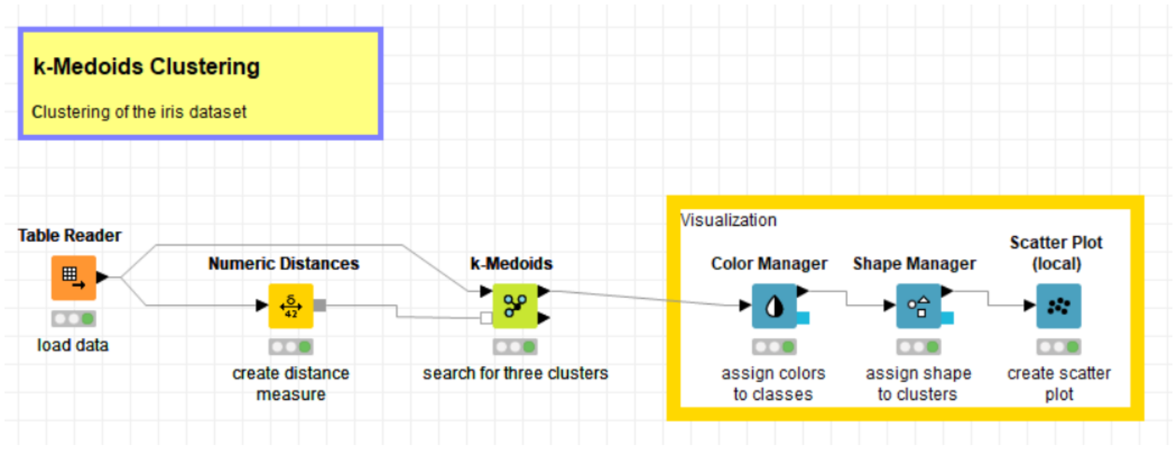
11.1 K-means clustering

KNIME: Task02_k-Means_Clustering



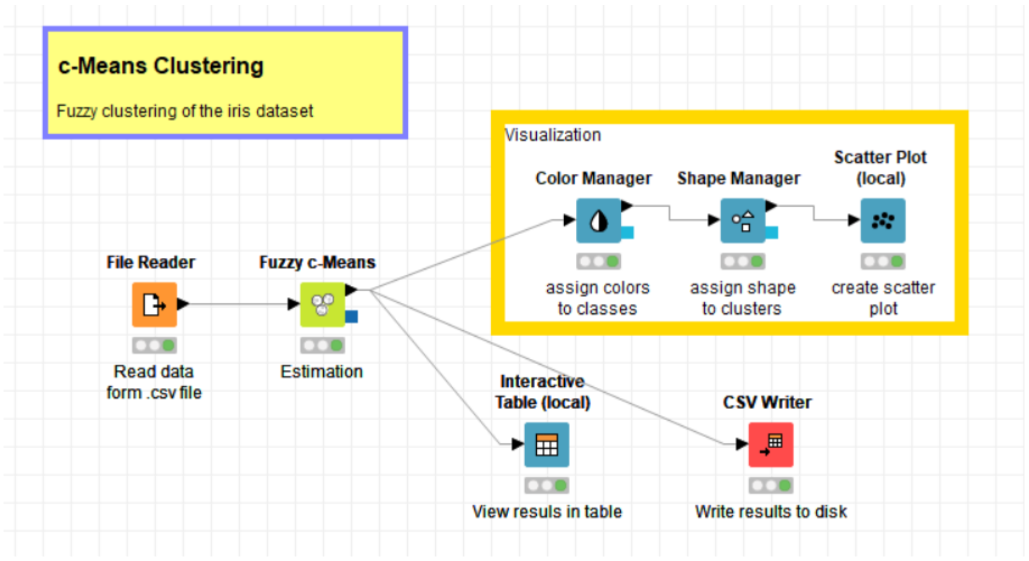
11.2 K-medoids clustering

KNIME: Task03_k-Medoids_clustering



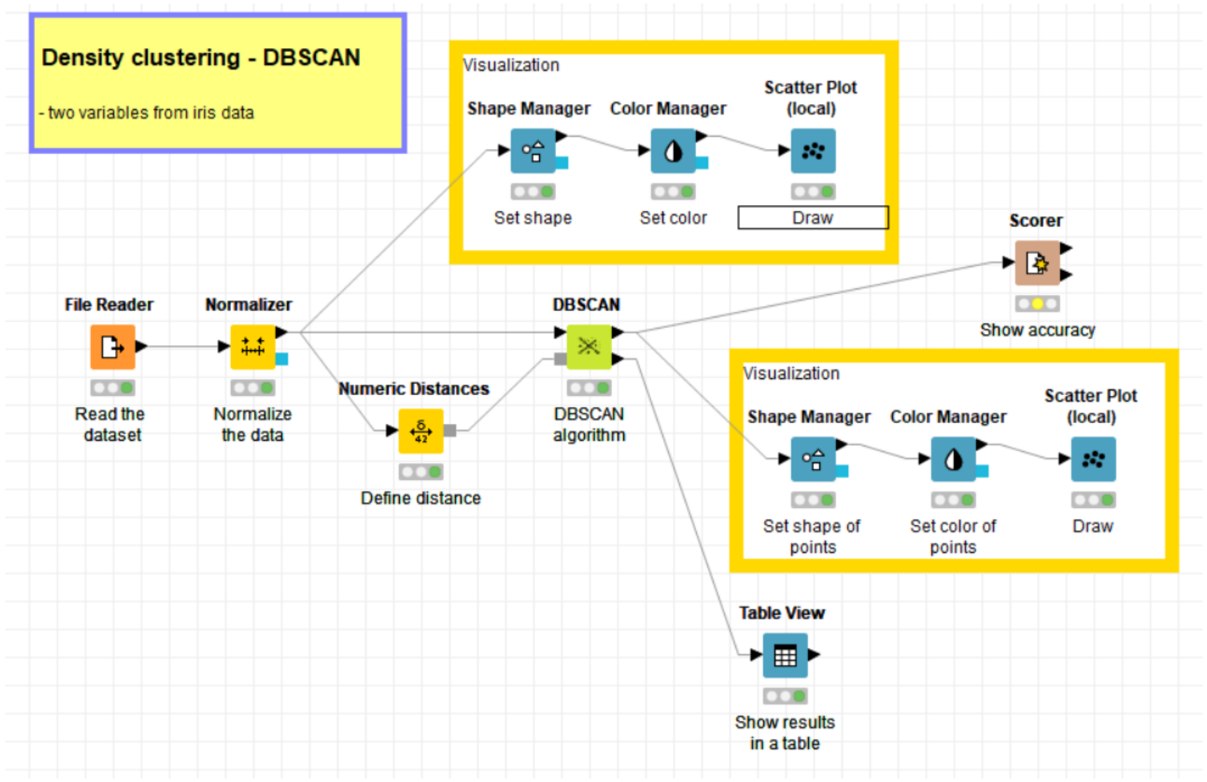
11.3 C-means clustering

KNIME: Task04_c-Means_Clustering



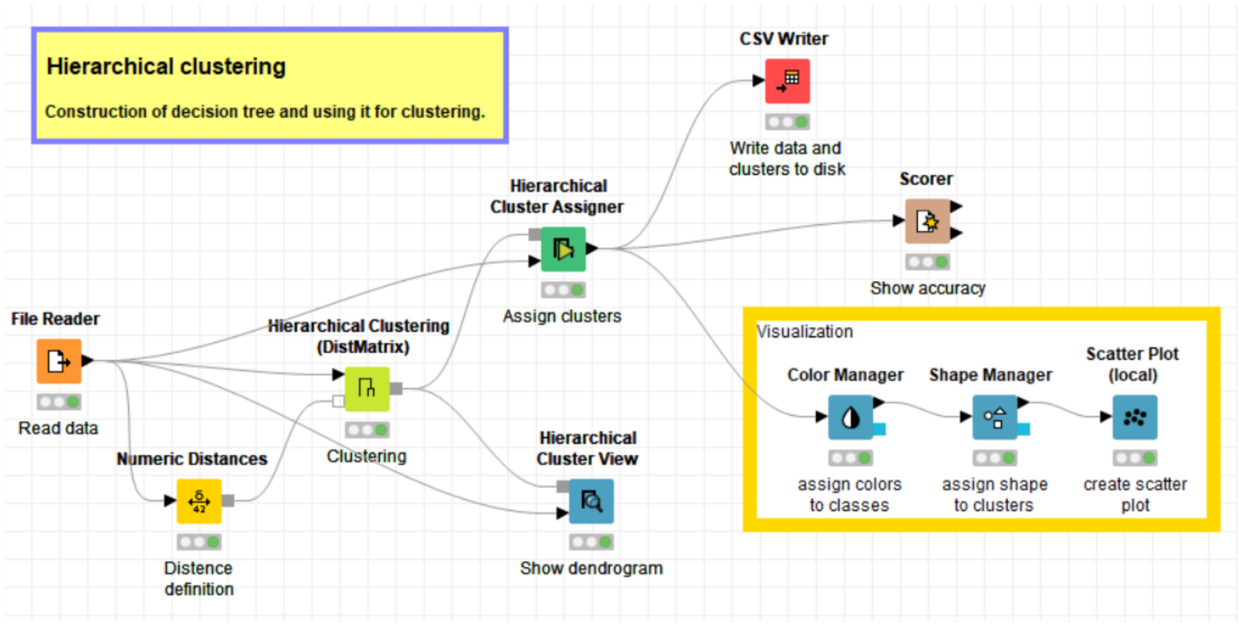
11.4 DBSCAN - density based clustering

KNIME: Task05_Density_Clustering



11.5 Hierarchical clustering

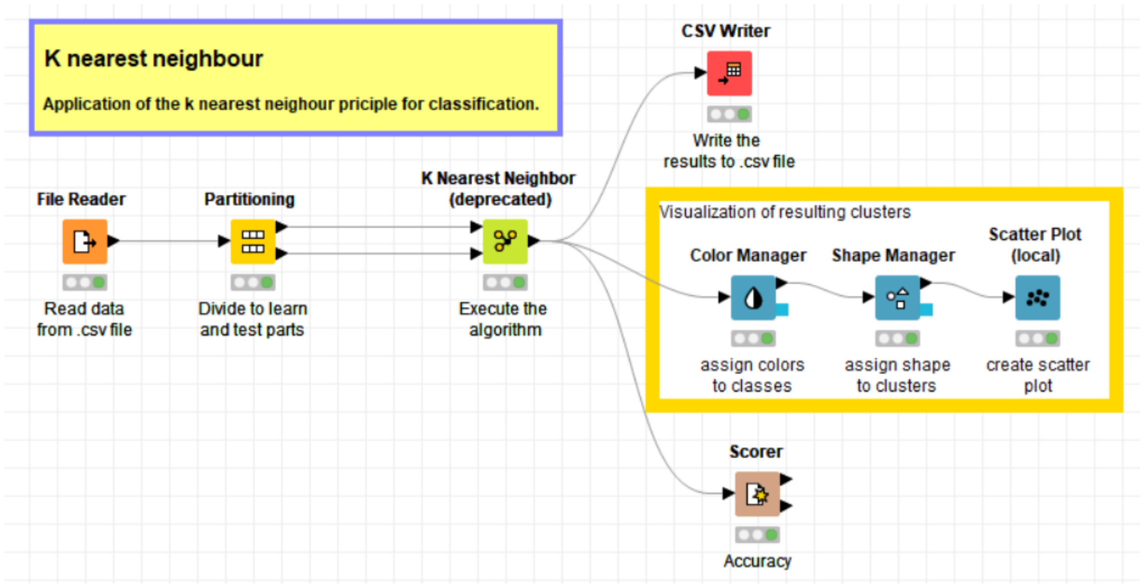
KNIME: Task06_Hierarchical_Clustrig



12 Classification

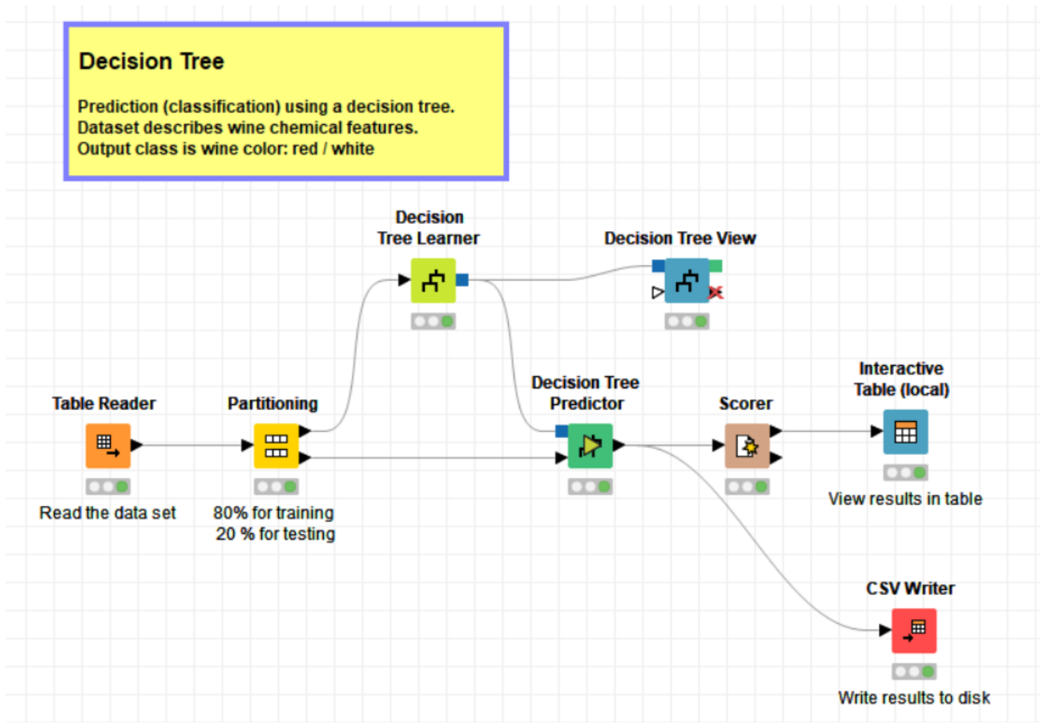
12.1 K-nearest neighbour classification

KNIME: Task07_k-NearNeighb



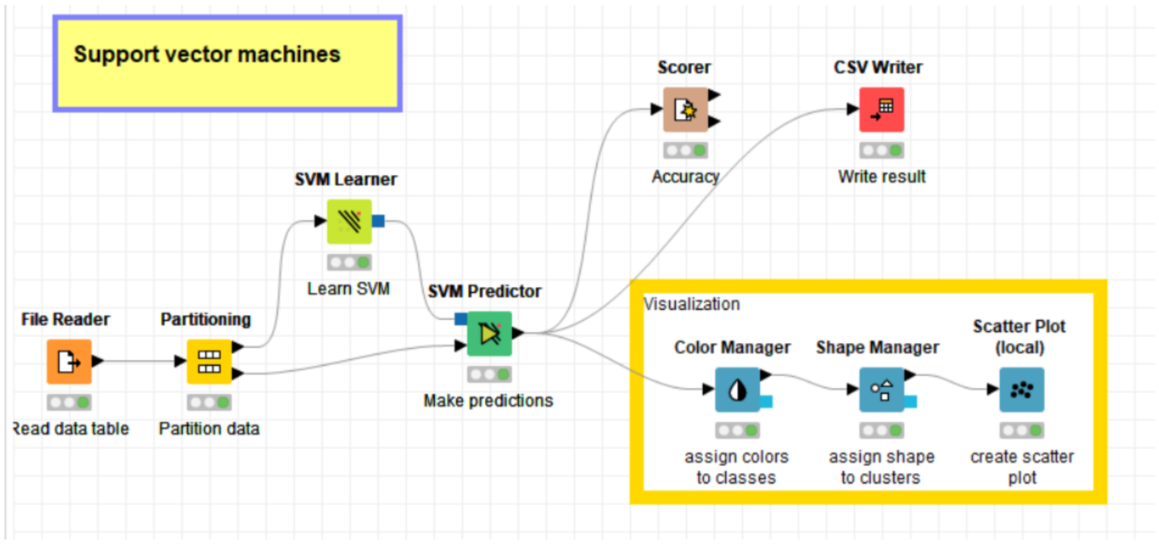
12.2 Decision tree classification

KNIME: Task08_Decision_Tree



12.3 Support vector machine classification

KNIME: Task09_Support_Vec_Mach


Support vector machines
Scorer
Accuracy
CSV Writer
Write result
SVM Learner
Learn SVM
File Reader
Read data table
Partitioning
Partition data
SVM Predictor
Make predictions
Visualization
Color Manager
assign colors to classes
Shape Manager
assign shape to clusters
Scatter Plot (local)
create scatter plot