Mathematical Methods for Data Analysis

Ivan Nagy

Faculty of Transportation Sciences
Czech Technical University, Prague

Contents

Part I

Clustering and Classification

1 Stochastic models

Model definition

Mathematical model is a formula (function) that assigns an output (result) to an input (argument). We can also speak about a cause and an effect and we will denote them x and y. Then the model takes form

$$y = g(x) \tag{1.1}$$

where $g(\cdot)$ is the function that assigns y to a given x. This model is **deterministic**. It means that to each value x it assigns always the same value of y.

If the process being modeled is under uncertainty the model takes the form

$$y = g(x) + e \tag{1.2}$$

where e is a noise affecting the output y. Then the model is called **stochastic**.

Remark

The uncertainty in the model may not be due to noise, but it is not wrong to think of uncertain variables as those that are deterministic but corrupted by noise.

The model (1.2) gives the output in the form of random variable with the expectation $f(x)$ and the distribution given by that of the noise (mostly of the normal form). All in all, we can define the model by the distribution of the output it produces. That is

$$f(y|g(x))$$

which is conditional distribution for given (known) value of $g(x)$.

Remark

Known $g(x)$ means that we must know not only x but also the function $g(\cdot)$ which is mostly described in a parametric form. Than we speak about "measured x" and "known model parameters".

Mostly the model acts in time. We always use a discrete time t which expresses number of period of measurement elapsed. That is

$$\tau = tT, \;\; t = 0, 1, 2, \cdots$$

where τ is continuous time (with the beginning at the start of the modeled action), T is the period of sampling (connected with the action) and t is the discrete time.

Then the random variables x and y become random processes, which are sequences of random variables with the notation x_t and y_t. The model then is

$$f(y_t|g(x_t)).$$

Types of models

Now, the function in the condition is too general. Most frequently it is defined in one of the following forms.

1. Constant k. The model is then
$$y_t = k + e_t \tag{1.3}$$
and it corresponds to measurement of constant variable with a noise.

2. Linear function of explanatory variables x
$$y_t = b_1 x_{1;t} + b_2 x_{2;t} + \cdots + b_m x_m + k + e_t \tag{1.4}$$
where b_i, $i = 1, 2 \cdots m$ are constant coefficients, x_i are entries of x_t, k is a constant (to cover the case if $y_t \neq 0$ for $x_t = 0$), e_t is the noise with zero expectation and constant variance r.

3. Linear dynamic function
$$y_t = b_0 u_t + a_1 y_{t-1} + b_1 u_{t-1} + \cdots + a_n y_{t-n} + b_n u_{t-n} + k + e_t \tag{1.5}$$
where a_i, b_i are regression coefficients and u is input variable (if its values can be set, then it can be considered a control).

The models listed above concern models of continuous output y_t. If y_t is a discrete random process, i.e. it can possess only limited number of different values (e.g. 0 or 1), then a different type of modeling must be adopted. In the case when both x_t and y_t are discrete, the described system has only finite number of states. They are all combinations of the couples $[x_t, y_t]$ and they all can be assigned by their probabilities. Such a model i called categorical and for $x_t \in \{1, 2, 3\}$ and $y_t \in \{1, 2\}$ it can be given a form of the following table - it is the probability function
$$f\left(y_t | x_t, \Theta\right) \tag{1.6}$$
where Θ is a table

$x_t =$	1	2
$y_t = 1$	$\Theta_{1\|1}$	$\Theta_{1\|2}$
$y_t = 2$	$\Theta_{2\|1}$	$\Theta_{2\|2}$
$y_t = 3$	$\Theta_{3\|1}$	$\Theta_{3\|2}$

where $\Theta_{i|j}$ are conditional probabilities, i.e. $\sum_i \Theta_{i|j} = 1 \; \forall j$ (their sums over columns are equal to one).

A special class of models concern those for description of nonnegative variables. For continuous variables it can be e.g. **exponential model**
$$f\left(y_t | a\right) = a \exp\left(-a y_t\right), a > 0, y_t = 0, 1 \cdots \infty \tag{1.7}$$
and for the discrete case **Poisson model**
$$f\left(y_t | \lambda\right) = \exp\left(-\lambda\right) \frac{\lambda^{y_t}}{y_t!}, \lambda > 0, y_t = 0, 1 \cdots \infty \tag{1.8}$$
or **binomial model**
$$f\left(y_t | p\right) = \binom{N}{y_t} p^{y_t} \left(1 - p\right)^{N - y_t} p \in (0, 1), y_t = 1, 2 \cdots N \tag{1.9}$$

Estimators

Constant model (1.3)

Here k is the average of measured y_t, $t = 1, 2, \cdots, N$

$$\hat{k}_N = \frac{\sum_{i=1}^{N} y_i}{N}$$

or with on-line update of statistics S (sum) and κ (number)

$$S_t = S_{t-1} + y_t$$

$$\kappa_t = \kappa_t + 1$$

with $S_0 = 0$ and $\kappa_0 = 0$. Then

$$\hat{k}_N = \frac{S_N}{\kappa_N}.$$

Static model (1.4)

Using least square method: Construct

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}, \quad X = \begin{bmatrix} x_{1;1} & x_{2;1} & \cdots & x_{m;1} & 1 \\ x_{1;2} & x_{2;2} & \cdots & x_{m;2} & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{1;N} & x_{2;N} & \cdots & x_{m;N} & 1 \end{bmatrix}$$

where m is the number of variables x and the estimate is

$$\hat{\theta}_N = (X'X)^{-1} X'Y$$

Dynamic model (1.5)

Define time interval $s = [n+1, n+2, \cdots N]$ and follow the previous example with

$$X = [u(s), y(s-1), u(s-1), \cdots, y(s-n), u(s-n), 1]$$

In $\hat{\theta}$ the parameters are stored in the way the matrix X is constructed.

Categorical model (1.6)

The statistics has the same matrix form as the parameter Θ. The update starts with $\Theta = 0$ and each time after measuring data x_t and y_t we add one to the entry $\Theta_{[y_t, x_t]}$.

The estimates are obtained by normalization of the statistics columns to th sum equal to one

$$\hat{\Theta}_{i,j} = \frac{S_{i,j}}{\sum_k S_{k,j}}, \forall j$$

Exponential model (1.7)

Here a is the inverse expectation of y. So $\hat{a} = \left(\frac{\sum y}{N}\right)^{-1}$ or on line

$$S_t = S_{t-1} + y_t$$

$$\kappa_t = \kappa_t + 1$$

with $S_0 = 0$ and $\kappa_0 = 0$. Then

$$\hat{k}_N = \frac{k_N}{S_N}.$$

Poisson model (1.8)

The parameter λ is the expectation, so its estimate is the average (see th constant model).

Binomial model (1.9)

The parameter is N times the expectation. So, the statistics for constant model (1.3) can be used and the final estimate is

$$\hat{p}_N = \frac{S_N}{N\kappa_N}$$

2 Models for clustering and classification

One of the prominent approaches in data mining is based on data modeling. The model describes density of data points in the data space (detects dense areas called clusters) and gives a possibility to detect the cluster to which a newly measured data record (point in the space) belongs.

First we will inspect models connected with this area and derive simple but general clustering and classification tool which is near to estimation of mixture models.

Then we will show the same procedure as before but with a simplifying assumption of independence of variables in the regression vector. This approach is known as Naive Bayes.

In the end, we will extend the previous attitude for models with unknown parameters. Then the parameters must be estimated from data. The estimation can be one shot or recursive (on-line).

2.1 Bayesian approach

Our approach to clustering and classification relies primarily on the use of modeling. Generally, we consider a data space X of finite vectors $x = [x_1, x_2, \cdots, x_n]$. These vectors represent points in the data space and we suppose, these points are somehow grouped with respect to their density (or spatial probability of occurrence). These groups of data vectors (represented as points in data space) are called components (classes). Our task is t0

1. detect these groups in the data space (clustering),
2. decide, which class a newly measured vector belongs to (classification).

The groups are labeled - each of them has its own flag (value of pointer). In our case, the flags will be integers $c = 1, 2, \cdots, n_c$. In classification, we measure a vector x and want to classify it. As the true class to which the vector belongs in unknown, the pointer c will be described by a discrete random variable with its probability function $f(c|x)$ where x is the vector to be classified.

Individual groups have their models $f(x|c) = f_c(x)$ which is a probabilistic description of vectors x belonging to the class c.

Putting it together

As we have seen, in connection with the tasks of clustering and classification, we have two models (for now, with known parameters).

Models of data (components)

$$f(x|c)$$

Model of classification (pointer)

$$f(c|x)$$

These two models are connected via Bayes rule

$$f(c|x) = f(x|c)\,\frac{f(c)}{f(x)}$$

following from the relation

$$f(c,x) = f(c|x)\, f(x) = f(x|c)\, f(c)$$

Remark

Naturally, from the first pdf in the Bayes rule, x is the cause and c is the effect. The Bayes rule says, that from the knowledge of the effect we can say something about the cause.

We will demonstrate these models in the following example.

Example

Let the joint model $f(c,x)$ be described trough the conditional pdf

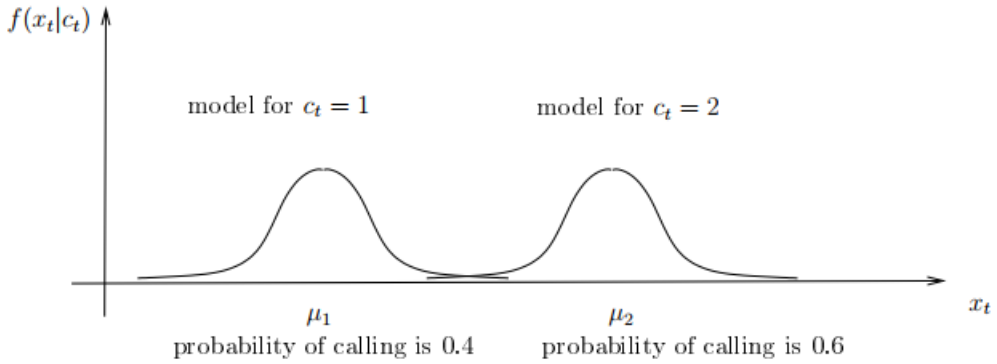
$$f(x|c=1) = f_1(x) = N_x(\mu_1, 1) \tag{2.1}$$

$$f(x|c=2) = f_2(x) = N_x(\mu_2, 1) \tag{2.2}$$

and the marginal probability function $f(c)$ given by the table

c	1	2
$f(c)$	0.4	0.6

Notice that each model is of a different type. The data are continuous, so their model type is also continuous (regression model), while the pointer is discrete and its description is a discrete probability function. As the data model depends on the pointer, we have to define two regression models - for each pointer value one model. They differ in parameters. The first component has expectation μ_1 while the second μ_2. The model can be demonstrated in the picture



The left part of the figure shows the model of the first component, which is active (generates data) in 40% of the model calls, and the right part shows the model of the second component, active in 60% of the cases.

Now, the joint model $f(c,x) = f(x,c)$, is given by a product of the conditional data model and the marginal pointer model; it is

$$f(x,c) = f(c)\, f(x|c) = \begin{cases} 0.4 N_x(\mu_1, 1) & \text{for } c = 1 \\ 0.6 N_x(\mu_2, 1) & \text{for } c = 2 \end{cases}$$

This is how a mixed model (i.e. model with both continuous and discrete variables) can be expressed.

Having the joint distribution of the model, we can express arbitrary conditional or marginal model. We already have the conditional model of the data $x|c$ and marginal one for the pointer c - we have defined them above. Now, we are going to determine the remaining two models.

Marginal data model

Marginal model for data x is obtained by summing the joint model over all values of the pointer, i.e. for $c = 1, 2$. We get

$$f(x) = 0.4N_x(\mu_1, 1) + 0.6N_x(\mu_2, 1)$$

which is a weighted sum of two Gaussian distributions.

Classification model

Can be computed as

$$f(c|x) = \frac{f(c,x)}{f(x)} \text{ or } f(x|c)\frac{f(c)}{f(x)} \propto \underbrace{f(x|c)}_{\text{component model}} \underbrace{f(c)}_{\text{component prior}} \tag{2.3}$$

$$f(c|x) = \begin{cases} \frac{0.4N_x(\mu_1,1)}{0.4N_x(\mu_1,1)+0.6N_x(\mu_1,2)} & \text{for } c = 1 \\ \frac{0.6N(\mu_2,1)}{0.4N_x(\mu_1,1)+0.6N_x(\mu_1,2)} & \text{for } c = 2 \end{cases} \propto$$

$$\propto \begin{cases} 0.4N_x(\mu_1, 1) & \text{for } c = 1 \\ 0.6N_x(\mu_2, 1) & \text{for } c = 2 \end{cases}$$

which is obvious (conditional pdf is proportional to the joint one), however, this is very important result claiming that:

Result: *The probability that x is to be classified into the class c is proportional to the value of the model of this component with the measured vector x inserted.*

Classification algorithm

The classification can run like this:

1. Measure new data vector x ($t > N$, where N is the length of learning).
2. Compute values of all component models with inserted vector x_t multiplied by prior probabilities of the components.
3. Classify the vector x_t to the component corresponding to the greatest computed value.

Remark

This holds for known models of components and pointer. If the parameters of these models are unknown, they have to be estimated and their point estimates can be used instead of the true parameters. It is an approximation but very good one. We will tackle this problem in more details later.

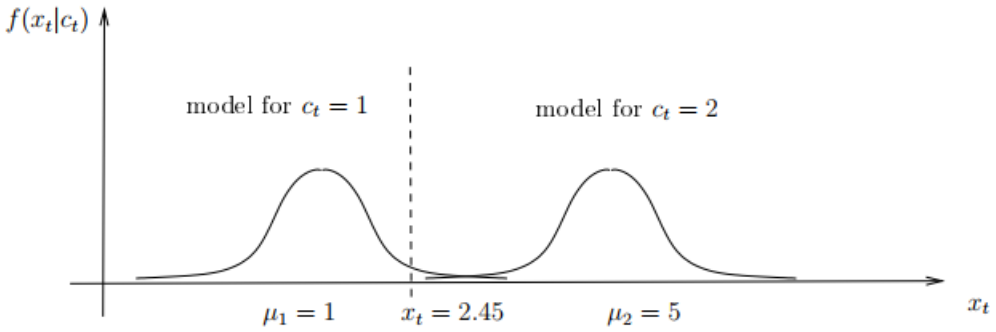
Now, we shall demonstrate the algorithm for the following numerical setting of the previous our example.

Let $\mu_1 = 1,\ \mu_2 = 5$ and the measured vector $x_t = 2.45$. The the values of the component models are

$$f_1 \propto 0.4 \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(2.45-1)^2\right\} = 0.056$$

$$f_2 \propto 0.6 \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}(2.45-5)^2\right\} = 0.009.$$

The first value is greater, the point $x_t = 2.45$ belongs to the first component. The situation in this simple example is clearly visible from the following picture



The point x_t lies closer to the first model, so the value of the model in x_t is greater.

Remark

The influence of the pointer model is not so important and is neglected in the picture. The main effect is caused by the component models and their values will be called proximity *as they express the closeness of the measured point to the centers of individual components.*

Clustering algorithm

The task of clustering consists in grouping the data into several classes according to given criterion. Bayesian model based clustering formulates the criterion as a closeness of the data to sub-models (components), which can be continuously estimated. The way of it is indicated in the following algorithm.

1. Set prior component models - e.g. for Gaussian components they are given by their centers and covariance matrices.
2. Subsequently measure data vectors from data space and
   (a) classify data into components
   (b) perform parameter estimation of component and pointer models. Estimation will be treated in the section with mixture models.
3. The resulting estimated components describe the found clusters.

2.2 Naive Bayes classification

This method is nothing but the previous case plus assumption of conditional independence of data variables, i.e. entries of the data vector x. With this assumption we have

$$f(x|c) = \prod_{i=1}^{n} f(x_{i;t}|c).$$

Remarks

1. *This formula can be explained by the assumed fact that the data in one cluster differ only by noise and thus are independent.*
2. *The independence brings considerable savings - instead of multidimensional model we can use only several one-dimensional ones. For normal components, instead of large covariance matrix we need only several (namely n) scalar variances.*

The pointer model (2.3) has now the form

$$f(c|x) \propto f(x|c) f(c) = f(c) \prod_{i=1}^{n} f(x_{i;t}|c)$$

where $f(x_{i;t}|c)$ are scalar models of individual variables within the class c.

2.3 Classification with learning

Now, let us take the previous case and add assumption of the ignorance of the true parameter values. These need to be learned (estimated) from data. Similarly as in the previous case, we will measure only one data vector x.

The models now will be:

Model of data in component c

$$f(x|c, \theta_c)$$

Model of pointer

$$f(c|\alpha_c)$$

Classification model

$$f(c|x)$$

Estimation model

$$f(\theta, \alpha|x)$$

As in the beginning of this section, we look for the classification pdf $f(c|x)$. To be able to construct it, we must introduce the model parameters θ and α. So, we start with pdf of all unknown objects $f(c, \theta, \alpha|x)$ and perform its factorization

$$f(c, \theta, \alpha|x) \propto \underbrace{f(x, c, \theta, \alpha)}_{\text{joint pdf}} = f(x|c, \theta, \alpha) f(c|\theta, \alpha) f(\theta, \alpha) =$$

$$= \underbrace{f(x|c,\theta_c)}_{\text{component model}} \underbrace{f(c|\alpha_c)}_{\text{pointer model}} \underbrace{f(\theta,\alpha)}_{\text{prior}}$$

where the first two pdfs are parameterized models of data and pointer, the last one is a prior description of parameters which is updated to posterior with the information carried by the data vector x. From this relation for the joint pdf we can obtain all needed

1. The classification

$$f(c|x) = \int_{\theta^*}\int_{\alpha^*} f(x,c,\theta,\alpha)\, d\alpha d\theta =$$

$$= \int_{\theta^*}\int_{\alpha^*} f(x|c,\theta_c)\, f(c|\alpha_c)\, f(\theta,\alpha)\, d\alpha d\theta \doteq \tag{2.4}$$

$$= \begin{cases} f\left(x|c=1,\hat{\theta}_1\right)\hat{\alpha}_1 & \text{for } c=1 \\ f\left(x|c=2,\hat{\theta}_1\right)\hat{\alpha}_2 & \text{for } c=2 \end{cases}$$

   where $f\left(x|c=1,\hat{\theta}_1\right) = N_x\left(\hat{\mu}_1,1\right)$, $f\left(x|c=2,\hat{\theta}_2\right) = N_x\left(\hat{\mu}_2,1\right)$, $\hat{\theta}_1 = \hat{\mu}_1$, $\hat{\theta}_2 = \hat{\mu}_2$, $\hat{\alpha}_1$, $\hat{\alpha}_2$ are point estimates of the prior parameters.

   Let the true parameters are the same as in the preceding example and let the prior information is expressed in these point estimates:

$$\hat{\mu}_1 = 2,\ \hat{\mu}_2 = 3,\ \hat{\alpha}_1 = .5,\ \hat{\alpha}_2 = .5$$

2. The estimation

$$f(\theta,\alpha|x) = \sum_c f(x,c,\theta,\alpha) =$$

$$= \sum_c \left[f(x|c,\theta_c)\, f(c|\alpha_c)\right] f(\theta,\alpha) \tag{2.5}$$

Example

Let us continue in our previous example, however, the parameters of the models will be unknown. The component models are (2.1) and (2.2) with unknown values of μ_1 and μ_2; the pointer model is

c	1	2
$f(c)$	α_1	α_1

with $\alpha_1,\ \alpha_2 \geq 0$ and $\alpha_1 + \alpha_2 = 1$.

Substitution to (2.4) gives

$$f(c|x) = \begin{cases} N_x\left(\hat{\mu}_1,1\right)\hat{\alpha}_1 = N(1,1)\,0.4 & \text{for } c=1 \\ N_x\left(\hat{\mu}_2,1\right)\hat{\alpha}_2 = N(5,1)\,0.6 & \text{for } c=2 \end{cases}$$

For measured $x = 2.45\cdots$

Here, the situation is formally the same as in the previous example.

The update of the estimated parameters will be

$$f(\theta, \alpha|x) = [N_x(\mu_1, 1)\alpha_1 + N_x(\mu_2, 1)\alpha_2] f(\mu, \alpha)$$

which is almost standard update for continuous and discrete models. The only difference is that the models are in a sum. Here, the summation form of the model does not matter but in recursive estimation we are in a serious trouble. As the Bayes rule is a product of pdfs and the model is a sum, its repetitive calling produces the posterior pdf in a form which gets more and more complex and its evaluation and storing in memory is unfeasible.

2.4 Classification with recursive learning

Herr we are going to tackle the general problem of on-line clustering and classification. The method steams from the cases described above and generalizes them to on-line estimation. It is based on mixture estimation and it performs parallel classification and clustering. For existing estimates of data clusters it classifies a newly measured data record (it determines weights - the probabilities of membership of the data record to individual clusters) and then used the data record for updating the cluster description, each with the corresponding weight.

As we deal with data in time, we introduce time indexes at variables.

The models now will be:

Model of data

$$f(x_t|c_t, \theta_{c_t})$$

Model of pointer

$$f(c_t|\alpha_{c_t})$$

Classification model

$$f(c_t|x(t))$$

Estimation model

$$f(\theta, \alpha|x(t))$$

where, we remind, $x(t) = [x_0, x_1, \cdots, x_t]$, x_0 denotes prior information.

For the derivation of the method, we will follow the last one.

$$f(c_t, \theta, \alpha|x(t)) \propto \underbrace{f(x_t, c_t, \theta, \alpha, x(t-1))}_{\text{joint pdf}} = f(x_t|c_t, \theta, \alpha) f(c_t|\theta, \alpha) f(\theta, \alpha|x(t-1)) =$$

$$= \underbrace{f(x_t|c_t, \theta_{c_t})}_{\text{comp. model}} \underbrace{f(c_t|\alpha_{c_t})}_{\text{point. model}} \underbrace{f(\theta, \alpha|x(t-1))}_{\text{prior}}$$

and here we come to a problem in estimation. Formally, it will be

$$f(\theta, \alpha|x(t)) \propto \sum_{c_t=1}^{n_c} [f(x_t|c_t, \theta_{c_t}) f(c_t|\alpha_{c_t})] f(\theta, \alpha|x(t-1))$$

as in the last case, but now, in recursive estimation due to the summation form of the model the computations are unfeasible. The posterior pdf does not preserve the form of the prior and gets more and more complex. Its remembering and evaluation overflows the possibility of standard computer abilities.

That is why we must use an approximation. The straightforward one would be to approximate the sum of pdfs by a single one of the same form as the prior pdf has. However, this is rather complex. So we will approximate by a trick.

Suppose, we know the true component to which the measured data record belongs. Then we can define a pointer

$$\delta\left(c_t, \hat{c}_t\right) = \begin{cases} 1 & \text{for } c_t = \hat{c}_t \\ 0 & \text{elsewhere} \end{cases}$$

where c_t is random variable and $\hat{c}_t$ its realization (the label of the true component). Thus, at each time instant t the pointer denotes the component that is really true (active - the data record x_t belongs to it). However, in reality, we do not know the active component. So, we must estimate the pointer as an expectation

$$E\left[\delta\left(c_t, \hat{c}_t\right)|x\left(t\right)\right] = \sum_{c \epsilon c_t} \delta\left(c, \hat{c}_t\right) f\left(c|x\left(t\right)\right) = P\left(c = \hat{c}_t|x\left(t\right)\right) \text{ for } c = 1, 2, \cdots, n_c$$

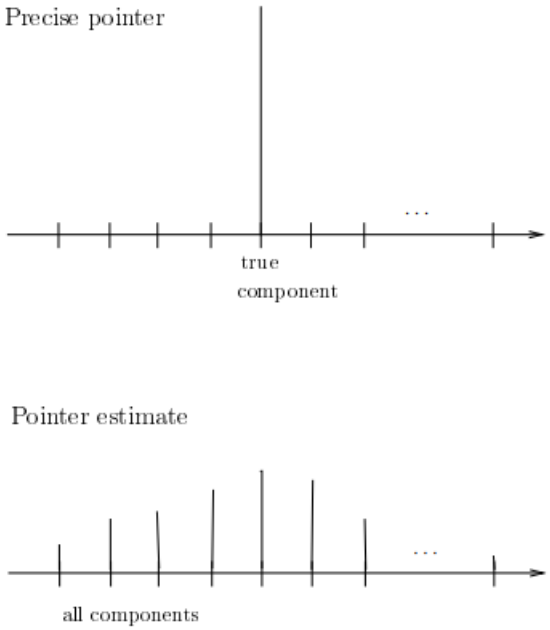
which is a vector of probabilities that the c-th component is active. We will call that vector actual components weights and denote it by $w_t = [w_{1;t}, w_{2;t}, \cdots, w_{n_c;t}]$ where

$$w_{i;t} = P\left(c_t = i|x\left(t\right)\right), \ i = 1, 2, \cdots, n_c$$

Remark

Notice that w_t depends on the actually measured data record x_t. It is the difference between it and the pointer model $f\left(c_t|\alpha\right)$. The pointer model expresses only historical knowledge about the activities of the component while w_t takes into account also x_t which is most important for the actual classification.

The effect of the approximation is following: Formerly, we needed to know the true active component. Now, we only need to know the probabilities that each individual component is active. The knowledge of the true active component is not required. It is like in the following picture



The pointer, now, is nothing but the classification pdf $f\left(c_t|x\left(t\right)\right)$. This is determined in the task of classification in the last section

$$f\left(c_t|x\left(t\right)\right) \underbrace{\propto}_{\text{Bayes}} f\left(x_t, c_t|x\left(t-1\right)\right) =$$

$$= \int_{\theta^*}\int_{\alpha^*} f\left(x_t, c_t, \theta, \alpha|x\left(t-1\right)\right) d\alpha d\theta =$$

$$= \int_{\theta^*}\int_{\alpha^*} f\left(x_t|c_t, \theta_{c_t}\right) f\left(c_t|\alpha\right) f\left(\theta, \alpha|x\left(t-1\right)\right) d\alpha d\theta =$$

or - using point estimates of parameters instead of full likelihoods

$$= f\left(x_t|c_t, \hat{\theta}_{c_t}\right) \hat{\alpha}_{c_t} f\left(\theta, \alpha|x\left(t-1\right)\right)$$

for measured x_t and $c_t = 1, 2, \cdots, n_c$.

Then, the estimation is performed in a standard way for given distribution of components and categorical distribution of the pointer model. The only difference is, that the update of statistics uses weighted data with just computed weights $w_t = f\left(c_t|x\left(t\right)\right)$.

Example

We will continue with the same example like in the preceding sections. We will:

1. Simulate a mixture with two static Gaussian components

$$f_1\left(x_t|\mu_1\right), \quad \mu_1 = 1$$
$$f_1\left(x_t|\mu_2\right), \quad \mu_2 = 5$$

with known variances equal to 1 and pointer model

$$f\left(c_t|\alpha\right), \quad \alpha = [0.4,\, 0.6]\,.$$

2. Estimate the mixture with initial parameters

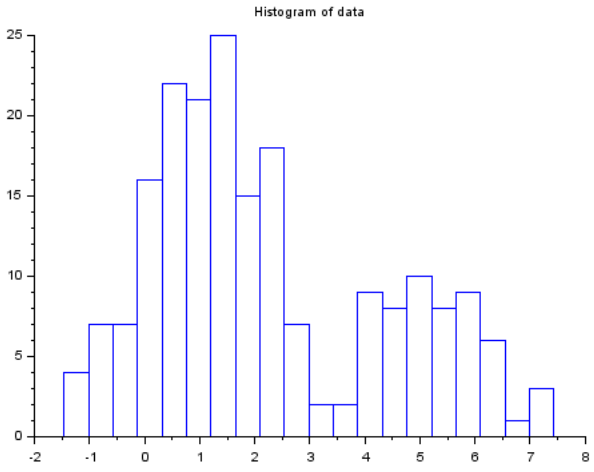
$$\hat{\mu}_{1;0} = 2, \quad \hat{\mu}_{2;0} = 3, \quad \hat{\alpha} = [0.5,\, 0.5]\,.$$

The program is here

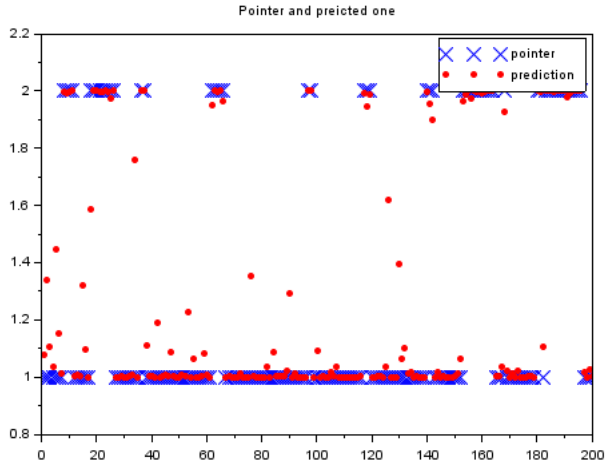
The program is described inside. Only some notes are necessary:

1. Simulation: first the pointer value is generated and according it a corresponding component is used for data generation.
2. The second part is estimation.
   (a) First, the initial parameters m and al are specified. K is the counter. Its initial value expresses the strength of prior information (the fictive number of data from which the information has been extracted).
   (b) Then, in the time loop, weights are computed. The computation is performed in logarithms, then it is roughly normalized by subtracting maximum, then exponent is taken and multiplication with al is performed and finally normalized to sum equal to one.
   (c) In the end of the loop, statistics are updated by weighted data and point estimates computed.

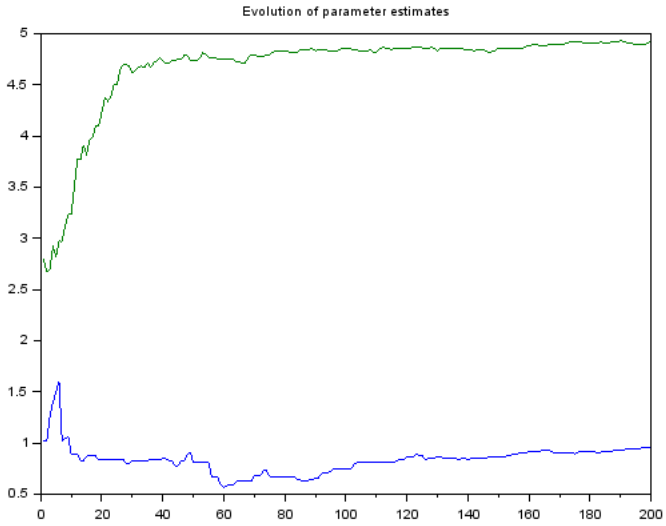
The results of estimation (classification) are in the following pictures



Here the histogram of data sample is plotted. It can be seen, that the components are slightly overlapping. The classification is not trivial.



Here, the simulated (blue) pointer values and the predicted (magenta) ones are plotted. The prediction is finally classified to the class which is closer to it. It can be seen that at the beginning, when the learning just started there are some errors. Gradually it improves and in the end all classifications are correct.



And here is supplementary information - evolution of expectation estimated during estimation. The initial estimates are gradually improved till they reach practically correct values (1 and 5).

Remark

The approach presented for the last time is practically equivalent to mixture estimation.

3 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.

3.1 Logistic regression

Model for variable c_t with Bernoulli distribution

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

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

The expectation of c_t is

$$E\left[c_t|p\right] = p$$

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*

$$\mathit{logit}\left(p\right) = \ln\left(\frac{p}{1-p}\right)$$

This logit will be modeled by regression

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

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

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

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

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

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

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

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

```
// DM_LogisReg.sce
// Example: Logistic regression with two independen variables
// ----------------------------------------
clc, clear, close, mode(0), warning('off')
getd _func
function LL=logLL(b,par)
  // log-likelihood of logistic regression
  x=par.x;                          // data x
  y=par.y;                          // data y
  Li=y.*(x*b)-log(1+exp(x*b));      // vector of log-models
  LL=-sum(Li);                      // log-likelihood
endfunction
function [f,g,ind]=fun(b,ind,par)
  // auxiliary function
  f=logLL(b,par);             // log-likelihood
  g=numderivative(logLL,b);   // gradient
endfunction

// SIMULATION ======================================================================
nd=200;                       // number of data
bS=[4 8 -1]';                 // simulated parameter
sd=1;                         // regression noise  z=x*b+sd*rand
x=[ones(nd,1) rand(nd,1,'n') 5-rand(nd,1,'n')];
z=x*bS+sd*rand(nd,1,'n');
p=exp(z)./(1+exp(z));
y=round(p);

// LOGISTIC REGRESSION ---------------------------------------
b0=[0 0 0]';                  // initial estimates of parameters (including ones)
par.x=x;                      // data x
par.y=y;                      // data y
```

```
// estimation
fce=list(fun,par);
[LLopt, b,  gopt, work, iters, evals, err]..
= optim (fce,b0,iprint=2,'ar',1e8,1e8);    // optimization
b,err

z=par.x*b;                      // regression
p=exp(z)./(1+exp(z));           // p=P(y=1|x)
yp=round(p);                    // rounding  <.5 ->0, >.5 -> 1

wrong=sum(y~=yp)                // number of wrong classification

// RESULTS
set(scf(),'position',[800 10 500 300]);
plot(1:nd,y,'bx',1:nd,yp,'r.')

Ep=variance(y-yp)/variance(y) // relative prediction error

scf();
plot(jiggle(y),jiggle(yp),'.','markersize',3)
title 'y against yp - jiggled'
```

3.2 Poisson regression

Model with Poisson distribution

$$f\left(c_t|\lambda\right) = \exp\left\{-\lambda\right\}\frac{\lambda^{c_t}}{c_t!} \tag{3.1}$$

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

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

Thus, for λ we have

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

The final model $f\left(c_t|\lambda\right)$ will be (3.1) with the above substitution - for log-likelihood we express directly its logarithm

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

Log-likelihood is

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

and it is maximized numerically.

Program to the Poisson regression is here

```
// DM_LogisReg.sce
// Example: Logistic regression with two independen variables
// ----------------------------------------
clc, clear, close, mode(0), warning('off')
getd _func
function LL=logLL(b,par)
  // log-likelihood of logistic regression
  x=par.x;                          // data x
  y=par.y;                          // data y
  Li=y.*(x*b)-log(1+exp(x*b));      // vector of log-models
  LL=-sum(Li);                      // log-likelihood
endfunction
function [f,g,ind]=fun(b,ind,par)
  // auxiliary function
  f=logLL(b,par);             // log-likelihood
  g=numderivative(logLL,b);   // gradient
endfunction

// SIMULATION ======================================================================
nd=200;                       // number of data
bS=[4 8 -1]';                 // simulated parameter
sd=1;                         // regression noise  z=x*b+sd*rand
x=[ones(nd,1) rand(nd,1,'n') 5-rand(nd,1,'n')];
z=x*bS+sd*rand(nd,1,'n');
p=exp(z)./(1+exp(z));
y=round(p);

// LOGISTIC REGRESSION ---------------------------------------
b0=[0 0 0]';                  // initial estimates of parameters (including ones)
par.x=x;                      // data x
par.y=y;                      // data y

// estimation
fce=list(fun,par);
[LLopt, b,  gopt, work, iters, evals, err]..
= optim (fce,b0,iprint=2,'ar',1e8,1e8);    // optimization
b,err

z=par.x*b;                    // regression
p=exp(z)./(1+exp(z));         // p=P(y=1|x)
yp=round(p);                  // rounding  <.5 ->0, >.5 -> 1

wrong=sum(y~=yp)              // number of wrong classification

// RESULTS
set(scf(),'position',[800 10 500 300]);
plot(1:nd,y,'bx',1:nd,yp,'r.')

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

```
scf();
plot(jiggle(y),jiggle(yp),'.','markersize',3)
title 'y against yp - jiggled'
```

4 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.

4.1 K-means algorithm

Let us have a data sample $X = [x_1, x_2, \cdots, x_N]$ where $x_t = [x_{1;t}, x_{2;t}, \cdots, x_{n;t}]$ is a data record (point) and N is total number of data records. 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. I yes, go to 1. If not, the algorithm ends.

Program

```
// DM_kmeans.sce
// K-means
// ----------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
endfunction

// SIMULATION
m=list();
m(1)=[0 1]';
m(2)=[5 2]';
m(3)=[3 8]';
n=[15 30 20]*10;
sd=1.5;
ny=length(m(1));
```

```
y=[];
for i=1:3
  for t=1:n(i)
    y=[y m(i)+sd*rand(ny,1,'n')]; // data generation
  end
end

// ALGORITM
nd=size(y,2);            // number of data
nc=length(m);            // number of clusters
yc=list(); cL=list();
// inicialization of centers
mi=min(y,'c');
ma=max(y,'c');
for j=1:nc
  C(j).c0=(mi+ma)/2+rand(ny,1,'n');     // initial centers of clusters
  C(j).c=C(j).c0;             // first centers are initial ones
end

for it=1:1000
  for j=1:nc
    C(j).cd=[];               // initialization of clusters
  end

  // distances of data from nodes
  for i=1:nd
    for j=1:nc
      d(j)=distance(C(j).c,y(:,i));     // distances of point from centers
    end
    [xxx,k]=min(d);           // minimal distance point from the k-th center
    C(k).cd=[C(k).cd y(:,i)];
  end

  df=0;
  for j=1:nc
    C(j).cs=C(j).c;           // remember centers from last step
    C(j).c=mean(C(j).cd,2);   // new centers
    df=df+sum(abs(C(j).c-C(j).cs));     // shift of centers
  end
  if df<.1
    break                     // end of iterations
  end
end

// RESULTS
k1=1:n(1);
k2=k1($)+1:k1($)+n(2);
k3=k2($)+1:k2($)+n(3);
set(scf(),'position',[800 200 600 400]);
```

```
// data
plot(y(1,k1),y(2,k1),'kd','markersize',12)
plot(y(1,k2),y(2,k2),'ks','markersize',12)
plot(y(1,k3),y(2,k3),'ko','markersize',12)
// clusters
plot(C(1).cd(1,:),C(1).cd(2,:),'r.','markersize',3)
plot(C(2).cd(1,:),C(2).cd(2,:),'b.','markersize',3)
plot(C(3).cd(1,:),C(3).cd(2,:),'g.','markersize',3)
title('Data and found clusters','fontsize',4)

disp(it,'number of iterations')
```

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.

4.2 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.
0. To each medoid find the points that are closest to it. They will be initial clusters.
0. Determine overall distance of points from their medians.
1. Randomly select one medoid and one non-medoid (data point that is not a medoid).
2. Swap them and again determine overall distance of points from their medians.
3. If the distance is smaller, continue by 1. If not, algorithm ends.

Program

```
// DM_cmedoids.sce
// c-medoids (simple - like genetic alg.)
// ---------------------------------------
```

```
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
endfunction
function d=distXY(X,Y)
  // Distance of vectors X and Y
  nX=size(X,2);
  nY=size(Y,2);
  d=zeros(nX,nX);
  for i=1:nX
    for j=1:nY
      d(i,j)=distance(X(:,i),Y(:,j));
    end
  end
endfunction
function dc=updateCls(md,s,u,y)
  // update of all distances after update of medoids
  // dc   distances points from individual medoids: matrix md X nd
  // md   nuber of clusters
  // s    indexes of medoids
  // u    indexes of non-medoids
  // construction of new clusters
  d0=distXY(y(:,s),y(:,u));   // distances between medoids and non-medoids
  [xxx,ic]=min(d0,'r');       // ic(k) is label of cluster
  c=list();                   //        to which y(:,k) belongs
  for j=1:md
    c(j)=find(ic==j);         // c(k) is vector of indxes of y
  end                         //        which belong to cluster k
  // evaluation of new clusters

  for j=1:md
    dc(j)=sum(distXY(y(:,s(j)),y(:,c(j))));
  end                         // sum of distances data from medoids
endfunction                   //        = optimality criterion
// ====================================================================

// SIMULATION
m=list();
m(1)=[1 1]';                  // data centers
m(2)=[5 2]';
m(3)=[3 8]';
sd=.5;                        // std of data
al=fnorm([1 3 2]);            // prababilities of modes
ny=length(m(1));              // dimension of y
```

```
nc=length(al);                // number of modes
nd=200;                       // length of data
md=3;                         // number of initial centers (points)

for t=1:nd
  i=sum(rand(1,1,'u')>cumsum(al))+1;
  y(:,t)=m(i)+sd*rand(ny,1,'n');   // data generation
end

// CLUSTERING - first step
s=samwr(1,md,1:nd);           // first medoids
u=setdiff(1:nd,s);            // first non-medoids
dc=updateCls(md,s,u,y);       // distances within initial clusters
d0=sum(dc);

dd=d0;
dd0=d0;
ss=s';

// CLUSTERING - iterations
for ite=1:1000
  s0=s;                       // remember medoids from last step
  u1=samwr(1,1,u);            // choice of one non-medoid
  s1=samwr(1,1,s);            // choice of one medoid
  // swap one medoid and one non-medoid
  s=setdiff(s,s1);
  s=[s,u1];                   // new medoids
  u=setdiff(1:nd,s);          // remaining non-medoids

  dc=updateCls(md,s,u,y);     // new distances in clusters
  d=sum(dc);

  if abs(d-d0)<.001           // test of end of iterations
    printf(' Počet kroků  %d\n\n',ite)
    break
  end

  if d<d0                     // test in the end of iteration (go on / go back)
    d0=d;                     // crit OK - remember its value
  else
    s=s0;                     // crit is not OK - go back to original medoids
  end

  // remember
  dd=[dd d];
  dd0=[dd0 d0];
  ss=[ss s'];
end
chk=[dd0;dd;ss];
```

```
// RESULTS
C=y(:,s);
scf();
plot(y(1,:),y(2,:),'.')
plot(C(1,:),C(2,:),'rx','markersize',12)
```

Program description

Function definition

- updateCls recomputes centers and evaluates the overall distance of points from medoids within individual clusters.

Simulation

Two dimensional data y are generated. nd is number of data, md is number of clusters.

Initialization

Select medoids, the rest of points are non-medoids. Compute the overall distance.

Iterations

Chose one medoid and one non-medoid. Swap them. Compute the overall distance and compare with the previous one. Check for the end.

4.3 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)

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \tag{4.1}$$

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

```
// DM_cmeans.sce
// c-means (fuzzy)
// Remark: weights are computed in the function CMupdt
// ----------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
endfunction
// ----------------------------------------
function d=distXY(X,Y,p)
  // Distance of vectors X and Y
  if argn(2)<3, p=1; end
  nX=size(X,2);
  nY=size(Y,2);
  d=zeros(nX,nY);
  for i=1:nX
    for j=1:nY
      d(i,j)=distance(X(:,i),Y(:,j),p);
    end
  end
endfunction
// ----------------------------------------
function [c,d]=CMupdt(c,y)
  // computation of weights and centers
  // c    clusters
  // y    data

  // distances  d
  d=distXY(c,y);               // distances of points and medoids

  // weights  u
  v=ones(d)./(d+1e-8);        // membership function
  u=fnorm(v,1);               // normoalization over clusters

  // centers  c
```

```
  un=fnorm(u,2);              // normalization over points
  for j=1:size(c,2)
    c(:,j)=y*un(j,:)';
  end
endfunction
// ----------------------------------------
function c=clusters(dn)
  // indexes of points for individual clusters
  // dn   normed distances
  // c    list of indexes of points for clusters
  [xxx,ic]=min(dn,'r');
  c=list();
  for j=1:size(dn,1)
    c(j)=find(ic==j);         // clusters
  end
endfunction
// ----------------------------------------

// SIMULATION
cS=list();
cS(1)=[1 1]';                           // centers for simulation
cS(2)=[5 2]';
cS(3)=[3 8]';
sd=.8;                                  // stdev of points
al=fnorm([1 3 2]);                      // prababilities of modes
ny=length(cS(1));                       // dimension of y
nc=length(al);                          // number of modes
nd=50;                                  // length of data
md=3;                                   // number of initial centers (points)

// SIMULATION
for t=1:nd
  i=sum(rand(1,1,'u')>cumsum(al))+1;
  y(:,t)=cS(i)+sd*rand(ny,1,'n');
end

// CLUSTERING - first step
p=2;                                    // distance  [(p-q)'*(p-q)]^(p/2)
j=fix(nd*rand(1,md,'u'))+1;             // indexes of initial centers
c=y(:,j);                               // initial centers
[c,d0]=CMupdt(c,y);                     // first update of centers
sd0=sum(d0);

// CLUSTERING - iteration
for ite=1:1000
  [c,d]=CMupdt(c,y);                    // new centers (medoids)

  sd1(ite)=sum(d);
  if abs(sd1(ite)-sd0)<.001             // test of the end of iterations
```

```
    printf(' Počet iterací  %d\n',ite)
    break
  end
  sd0=sd1(ite);
end

cL=clusters(d);                          // indexes of points in clusters

// RESULTS
tx=['r.';'m.';'g.'];
scf();
plot(y(1,:),y(2,:),'x','markersize',7)
for j=1:md
  if ~(isempty(y(1,cL(j))) | isempty(y(2,cL(j))))
  plot(y(1,cL(j)),y(2,cL(j)),tx(j),'markersize',6)
  end
end
plot(c(1,:),c(2,:),'s')
```

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.

4.4 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" and "buff" (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 "cluss" and its neighborhood add to "buff".
5. From "buff" remove all points that have already been used (those that are in some cluster).
6. Repeat from 4. until "buff" is not empty. Otherwise continue.
7. Remember the created cluster "clus" and prepare the variable for new one.
8. If there exists another free inner point, put it to "clus" and go to 4. If not, stop the algorithm.

Clusters are formed by points that are connected.

Program

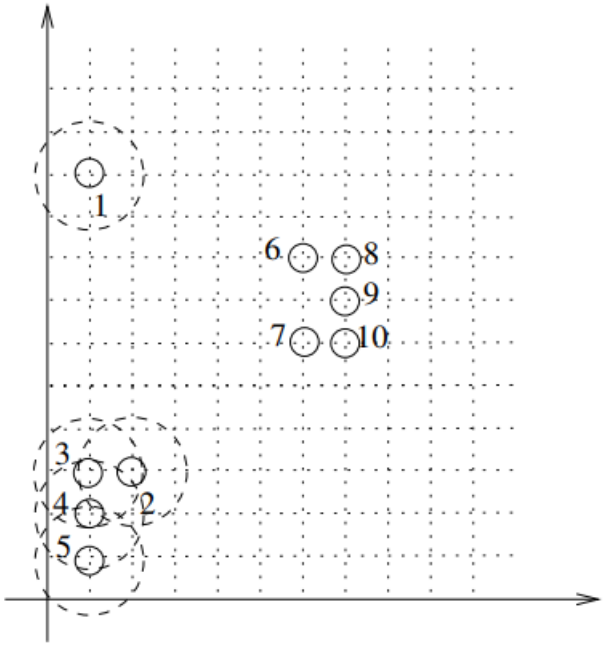
```
// DM_dbscan.sce
// Dbscan
// ----------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
endfunction
function b=board(x)
  // boards for graph
  b=[min(x(1,:))-.2 max(x(1,:))+.2 min(x(2,:))-.2 max(x(2,:))+.2];
endfunction

// SIMULATION
p=[.1 .2 .1 .4 .2];                // switchin parameter
th=[0 0; 0 3; 1 2; 2 1; 3 3]';     // centers
nd=100;                            // number of data
for i=1:nd
  j=sum(randu(1,1)>cumsum(p))+1;
  x(:,i)=.3*randn(2,1)+th(:,j);
end
bo=board(x);

// CLUSTERING
```

```
eP=.5;                              // radius of neighbourhood
mP=3;                               // minimum of points

// marking of inner points
V=[];                               // inner points
X=list();                           // neighbourhood of inner points
for i=1:nd
  X(i)=[];
  for j=setdiff(1:nd,i)
    if distance(x(:,i),x(:,j))<eP
      X(i)=[X(i) j];                // indexes of neighbourhood
    end
  end
  if length(X(i))>=mP
    V=[V i];                        // inner points
  end
end

// creation of
C=list();                 // clusters
b=V(1);                   // auxiliary variable
M=[];                     // already used points
k=1;                      // label of actual cluster
for h=1:100               // cycle for various clusters
  CC=[];                  // actual cluster
  while ~isempty(b)       // cycle for one cluster
    b1=b(1);              // one inner point
    CC=[CC b1];           // new point to cluster
    b=union(b,X(b1));     // add neighbourhood to b (auxiliary var.)
    b=setdiff(b,CC);      // removing just used point from b
  end
  if isempty(CC)
    break                 // end of algorithm
  end
  M=[M CC];               // remembering points from a cluster
  Vr=setdiff(V,M);        // inner points that are still not used
  if 1
    C(k)=gsort(CC,'g','i'); // actual cluster(with border)
  else
    C(k)=intersect(V,CC);   // actual cluster(without border)
  end
  k=k+1;                  // next cluster
  b=Vr(1);                // still not used point -> b
end
nC=length(C);             // number of clusters

// RESULTS
tx=['.r';'.b';'.g';'.m';'.k'];
set(scf(),'position',[600 100 900 400])
```

```
subplot(121)
plot(x(1,:),x(2,:),'c.')           // data
set(gca(),'data_bounds',bo)
title Data
subplot(122)
for i=1:nC
  plot(x(1,C(i)),x(2,C(i)),tx(i))  // clusters
end
set(gca(),'data_bounds',bo)
title Clusters
```

Example

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.

4.5 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. Its level is equal to the number of points in joined clusters.
3. The coordinates of the cluster lie on a connecting line of the coordinates of clusters to be joined in the proportion of their levels (the higher level the nearer).
4. Remember the clusters from which the new one has been created (hierarchy).
5. Repeat from 2 until only one cluster remains.

Remarks

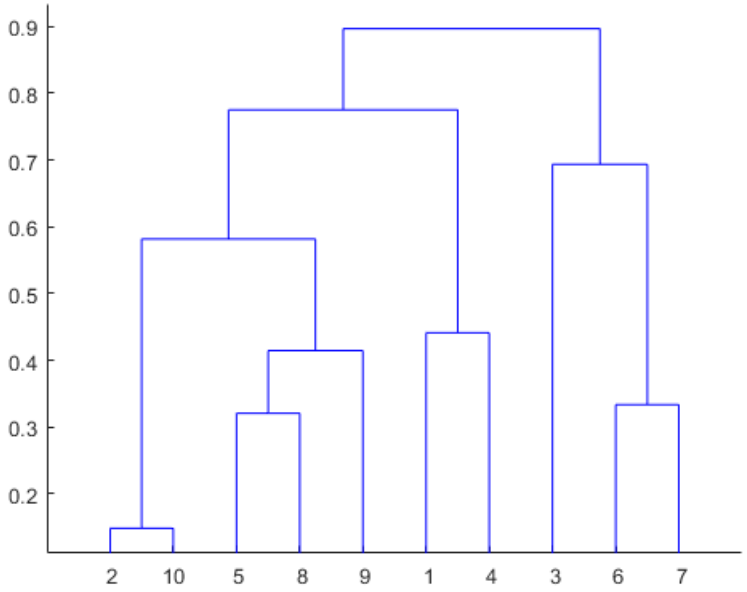
1. *The distance is Euclidean. It is computed between coordinates of clusters.*
2. *Coordinates of clusters on the level 1 are those of the points. For clusters generated by joining clusters with coordinates with levels h_i a h_j are coordinates given as follows:*
   The line connecting coordinates of the two clusters is

   $$x = x_i + t\,(x_j - x_i)\,,\ t \in (0,1)$$

   The point in the ratio of the levels (nearer to the cluster with higher level) is given by the parameter $t = \frac{h_j}{h_i+h_j}$. So

   $$x = x_i + \frac{h_j}{h_i + h_j}\,(x_j - x_i) = \frac{h_i x_i + h_j x_j}{h_i + h_j}$$

3. *Dendrogram is a special graph that shows the structure of hierarchical clustering as shown in the picture*



The resulting clusters can be determined on the basis of the dendrogram which can be drawn manually. The program gives the matrix C, where in each row the number of cluster, the distance of the coordinates of parents, and numbers of the parents can be found. The drawing will start in the cluster with the highest number (the last row of the matrix). In the graph, in the middle of the axis x and in the level of the distance (the second column in the matrix) on the axis y, we draw a circle and write a number of the cluster inside it. In the matrix C, find the parents of the node and draw the circles with their numbers in a corresponding levels on the axis y (the position on the axis x is arbitrary). We repeat this procedure until we exhaust all clusters that have been created by joining, only clusters with level one remain.

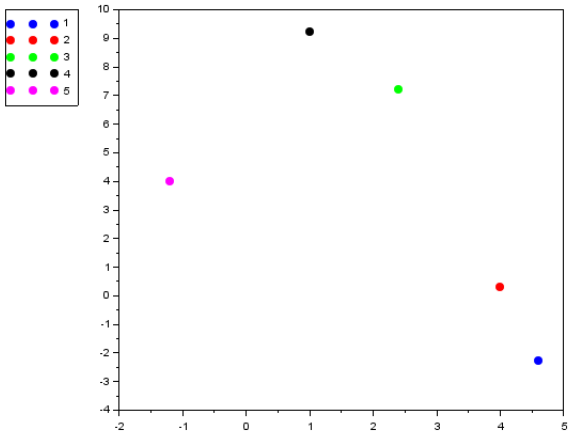
According to the desired number we can proceed as follows in determining the clusters :

We draw a horizontal line that intersect vertical lines of the dendrogram. The line can be shifted up or down. The number of intersections of the horizontal line with the vertical ones gives the number of created clusters. The points belonging to individual clusters are in the axes x below the intersection.

Example

We have 5 points

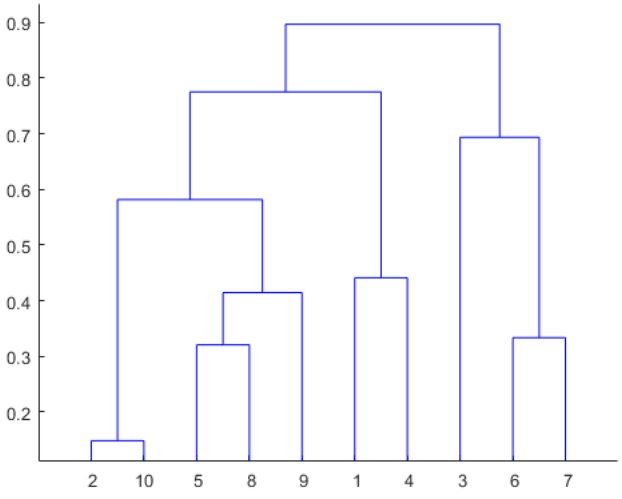
i	1	2	3	4	5
y_1	4.6	4.0	2.4	1.0	-1.2
y_2	-2.3	0.3	7.2	9.2	4.0



The matrix C is

$$C = \begin{bmatrix} 6, & 2.44, & 3, & 4 \\ 7, & 2.67, & 1, & 2 \\ 8, & 5.10, & 5, & 6 \\ 9, & 8.58, & 7, & 8 \end{bmatrix}$$

Construction of dendrogram starts with the last row (the cluster 9). We draw a circle in the middle of the axis x and in the height 8.58. Its parents are clusters 7 and 8. Those can be drawn to the left and right from the node 9 in heights 2.67 and 5.10. We continue in this way until we obtain the dendrogram according to the following picture



If we cut the dendrogram so that we obtain three intersections, we obtain the clusters

$$C_1 = \{1, 2\}\,,\ C_2 = \{5\}\,,\ C_3 = \{3, 4\}\,.$$

A comparison with the data plot confirms the clusters created.

Two clusters would be $C_1 = \{1, 2\}$ a $C_2 = \{3, 4, 5\}$.

Program

```
// DM_hierAgl.sce
// Hierarchical clustering (agglomerative)
// ---------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y)
  x=x(:); y=y(:);
  e=x-y;
  d=sqrt(e'*e);
endfunction
// =====================================================================

// SIMULATION ----------------------------------------------------------
m=list();
m(1)=[1 1]';                        // centers
m(2)=[5 2]';
m(3)=[3 8]';
sd=2.5;                             // stdev of points
al=fnorm([1 3 2]);                  // switching parametwr
ny=length(m(1));                    // number of variables
nc=length(al);                      // number of modes
nd=5;                               // number of data

for t=1:nd
  i=sum(rand(1,1,'u')>cumsum(al))+1;
  y(:,t)=m(i)+sd*rand(ny,1,'n');    // simulation
  c(1,t)=i;
end

// structre variable definition
for i=1:nd
  cL(i).y=y(:,i);   // data point or cluster
  cL(i).n=1;        // numb. of points in cluster
  cL(i).p=[];       // parents
  cL(i).v=[];       // distance
end

// ALGORITHM
nc=nd;
cc=1:nd;
for ite=1:(nd-1)
```

```
  lc=length(cc);
  d=zeros(lc,lc);
  for i=1:lc
    for j=1:lc
      if i<j
        d(i,j)=distance(cL(cc(i)).y,cL(cc(j)).y); // distances between points
      else
        d(i,j)=%inf; // symmetrical entries
      end
    end
  end

  // grouping points
  [v,ii]=min(d);    // nearest point
  i1=ii(1);
  i2=ii(2);

  nc=nc+1;
  n1=cL(cc(i1)).n;
  n2=cL(cc(i2)).n;
  y1=cL(cc(i1)).y;
  y2=cL(cc(i2)).y;
  cL(nc).y=(n1*y1+n2*y2)/(n1+n2);  // joining two points (clusters)
  cL(nc).n=n1+n2;
  cL(nc).p=cc(ii);
  cL(nc).v=v;
  cc(ii)=[];
  cc=[cc nc];
end

// determining clusters
C=[];
for i=(nd+1):(2*nd-1)
  C=[C; [i cL(i).v cL(i).p]];
end

// RESULTS
set(scf(),'position',[800 100 600 500])
tx=['.r';'.b';'.g';'.k';'.m';'.y'];
tn=['1','2','3','4','5'];
for i=1:nd
  plot(y(1,i),y(2,i),tx(i,:),'markersize',8)
end
legend(tn(1:nd),-2);

disp(' node  distance    p1   p2  (p = parents)')
disp(C)
```

Divisive clustering

In divisive clustering we proceed from top to bottom. We start with one cluster that contains all data points and subsequently we 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 - 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: to 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. We shift anti-medoid into the cluster C_L a compute the value of the criterion H_{LR}.
3. We try to add another point that is closer to the previously added.
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

```
// DM_hierDiv.sce
// Hierarchical clustering (divisive)
// ----------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
```

```
endfunction
// ----------------------------------------
function d=distXY(X,Y,p)
  // Distance of vectors X and Y
  if argn(2)<3, p=1; end
  nX=size(X,2);
  nY=size(Y,2);
  d=zeros(nX,nY);
  for i=1:nX
    for j=1:nY
      d(i,j)=distance(X(:,i),Y(:,j),p);
    end
  end
endfunction
function h=H1(cL,cR,y)
  // sum of mutual distances of points
  h=0;
  for i=cL
    for j=cR
      h=h+distance(y(:,i),y(:,j));
    end
  end
endfunction

// ====================================================================
// DATA
nd=120;
py=[.3 .5 .2];
th=[-2 6; 6 3; 8 15]';
cv=3;
for i=1:nd
  iy=sum(rand(1,1,'u')>cumsum(py))+1;
  y(:,i)=th(:,iy)+cv*rand(2,1,'n');
end

// ALGORITHM
c=list();
cL=1:size(y,2);                // data indexes
for itA=1:100                  // iterations between clusters
  cR=[];

  // initialization
  D=distXY(y(:,cL),y(:,cL));
  Da=sum(D,2);
  [xxx,cc1]=max(Da);             // cc1 - pointer to anti-medoid
  c1=cL(cc1);                    // c1  - index of anti-meoid
  ci=cc1;                        // ci  - storing of used clusters

  cL=setdiff(cL,c1);            // old (all)
```

```
  cR=union(cR,c1);              // new (is added)

  h2=H1(cL,c1,y);

  // iterations in one cluster
  for ite=1:100
    Dn=zeros(cL);
    for i=1:length(cL)
      Dn(i)=distance(y(:,c1),y(:,cL(i)));
    end
    [xxx,cc2]=min(Dn);
    c2=cL(cc2);
    ci=[ci cc2];                  // is added for trial use

    cL=setdiff(cL,c2);
    cR=union(cR,c2);
    h1=H1(cL,cR,y);

    c1=c2; cc1=cc2;

    if h1<=h2
      ci=setdiff(ci,cc2);         // if not used, it is removed
      break
    end
      h2=h1;
  end
  c(itA)=cR;
  if isempty(cL), break, end
end

// RESULTS
tx=['.r';'.b';'.g';'.k';'.m';'.y';'*r';'*b';'*g'];
set(scf(),'position',[800 100 600 500])
for i=1:length(c)
  plot(y(1,c(i)),y(2,c(i)),tx(i))
end
c
```

5 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.

5.1 K-nearest neighbour

It is a basic form of classification.

We have data $X = \{x_i\}_{i=1}^{N}$ with detected clusters. We can get them using some method of clustering. The task is: for a newly measured data point y, to assign it to some cluster.

The procedure of classification is 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

```
// DM_knearest.sce
// K nearest neighbour
// ---------------------------------------
clc, clear, close, mode(0)
getd _func
function d=distance(x,y,p)
  // Euclidean distance (for p=1)
  if argn(2)<3, p=1; end
  x=x(:); y=y(:);
  e=x-y;
  d=(e'*e)^(p/2);
endfunction
// ---------------------------------------
function d=distXY(X,Y,p)
  // Distance of vectors X and Y
  if argn(2)<3, p=1; end
  nX=size(X,2);
  nY=size(Y,2);
  d=zeros(nX,nY);
  for i=1:nX
    for j=1:nY
      d(i,j)=distance(X(:,i),Y(:,j),p);
    end
```

```
  end
endfunction
function tx=scfmark()
  // marks for plot
  tx=['.b';'.r';'.g';'.k';'.m';
      'xb';'xr';'xg';'xk';'xm';
      'db';'dr';'dg';'dk';'dm';
      'sb';'sr';'sg';'sk';'sm';
      '*b';'*r';'*g';'*k';'*m';
      'pb';'pr';'pg';'pk';'pm';
      '+b';'+r';'+g';'+k';'+m';
      'ob';'or';'og';'ok';'om'];
endfunction
function [h,f]=vals(a)
  // [h f]=vals(a)  find different values of a variable
  //                and their frequencies
  // h       values and frequencies  [vals;abs_freq]
  // f       relative frequencies

  a=a(:)';
  b=gsort(a,'g','i');
  [v,m]=unique(b);
  dm=diff(m);
  n1=length(b)+1;
  n=[dm n1-m($)];
  f=n/sum(n);
  h=[v(:)';n];

  if sum(n)~=max(size(a))
    disp('Error: in vals.sci')
    return
  end
endfunction
// ====================================================================

// SIMULATION
m=list();
m(1)=[1 1]';
m(2)=[5 2]';
m(3)=[3 8]';
sd=2.5;
al=fnorm([1 3 2]);
ny=length(m(1));
nc=length(al);
nd=130;

for t=1:nd
  i=sum(rand(1,1,'u')>cumsum(al))+1;
  y(:,t)=m(i)+sd*rand(ny,1,'n');   // data generation
```

```
  c(1,t)=i;
end

// ALGORITHM
k=15;                              // k nearest neighbour (this is k)
i=sum(rand(1,1,'u')>cumsum(al))+1;
z=m(i)+sd*rand(ny,1,'n');          // choice of a point
ic=i;

d=distXY(z,y);
[ds,j]=gsort(d,'g','i');
jk=j(1:k);                         // the nearest k points
ck=c(jk)
v=vals(ck);
[xxx,i]=max(v(2,:));
cz=v(1,i)

// RESULTS
tx=['.r';'.b';'.g';'.k';'.m';'.y';'*r';'*b';'*g'];
scf();
for j=1:length(m)
  i=find(c==j);
  plot(y(1,i),y(2,i),tx(j),'markersize',3)
end
legend('1','2','3');
plot(z(1),z(2),'ko','markersize',8)
```

5.2 Decision trees

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.

The principle of tree construction if following:

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 one 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.

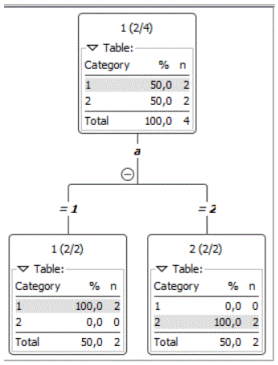
Example

Let us have the following data

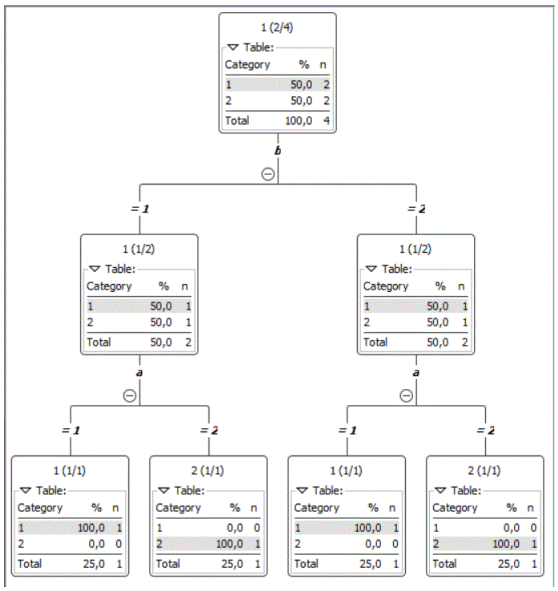
t	x_1	x_2	c
1	1	1	1
2	1	2	1
3	2	1	2
4	2	2	2

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



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

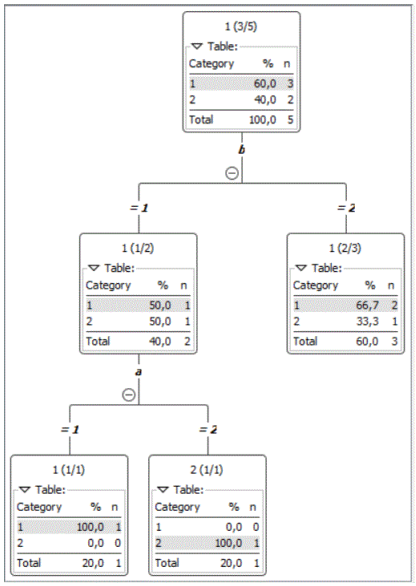


However, both the trees led to deterministic decision making (the final percent are 100%).

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



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.

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**

(5.1)

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

The values of variables are:

sex: M = man, Z = woman;

car: 0 - does not have, 1 - has

fare: L - low, S = medium, V - high;

way: B - bus, V - train, A - car.

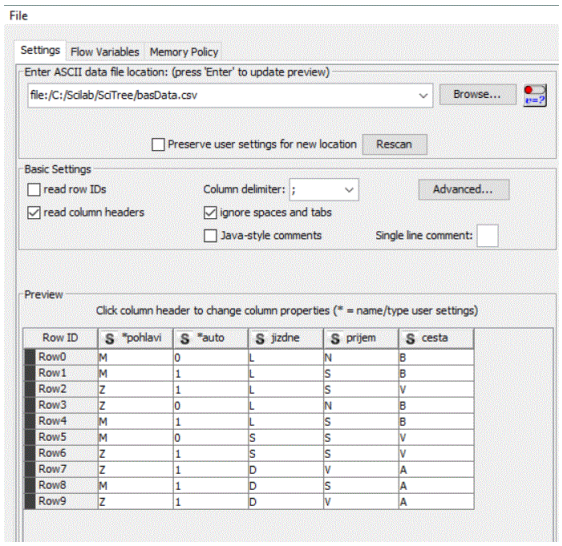
The task is to decide about the way (B, V, A) on the basis of the values in data records.

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.

```
pohlavi;auto;jizdne;prijem;cesta
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
```

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



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

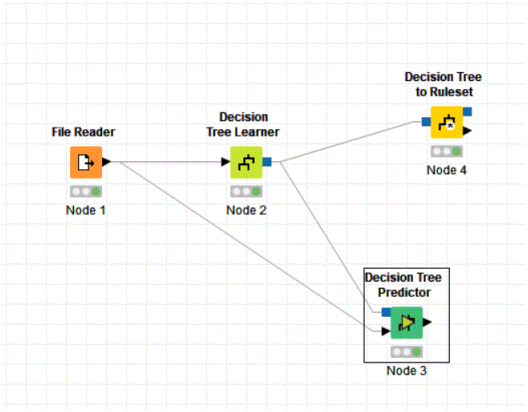
- The pointer variable must be set as string. The rest of variables can stay as they are.
- Strings are sorted by values the other by intervals.
- The change of the variable type can be done in the menu which can be obtained by clicking at the title of the variable in the data table below. After a click 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

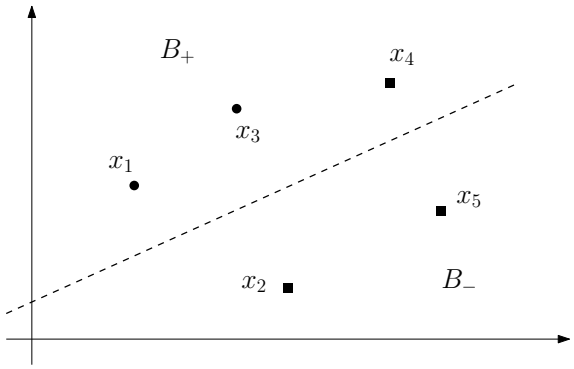
If the tree ends prematurely, it is necessary to set Number of records per node = 1 in the menu Configure in the tool Decision Tree Learner. It means that the decision rule can be derived from only one data record.

5.3 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 a 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

$$|w|\to\min$$

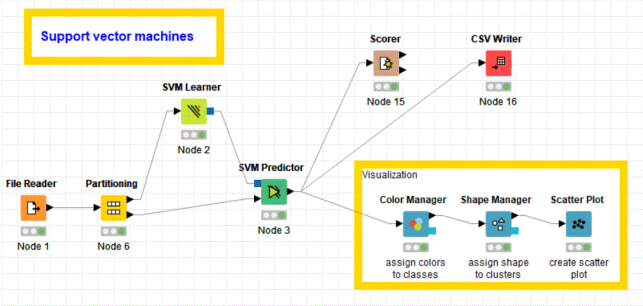
on condition that

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

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

Program KNIME

Create tho 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 after clicking on the task icon down in the menu.

2. The data file can be changed directly on disk. If there are new variables (not only values), it is necessary to perform new Configuration otherwise only to run Execute.

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

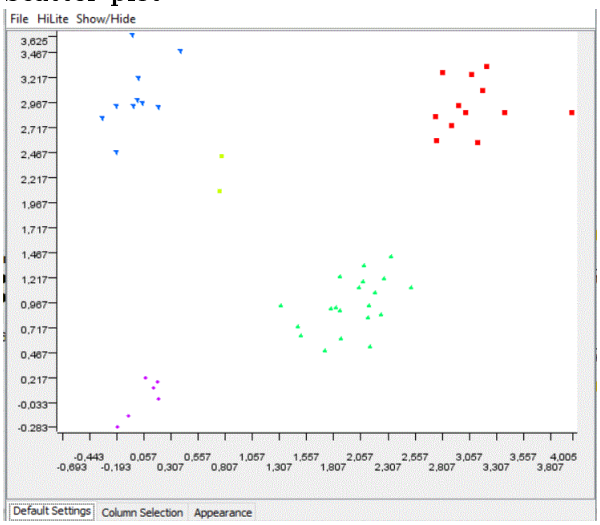


Table of classifications

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

Part II

Supplements

5.4 Bayes rule

Derivation

$$f(A,B|C) = \begin{cases} f(A|B,C)\, f(B|C) & \text{z jedné strany, nebo} \\ f(B|A,C)\, f(A|C) & \text{z druhé strany.} \end{cases}$$

By comparison of both right hand sides we get

$$f(A|B,C)\, f(B|C) = f(B|A,C)\, f(A|C).$$

From it

$$f(B|A,C) = \frac{f(A|B,C)\, f(B|C)}{f(A|C)}. \tag{5.2}$$

which can also be written as

$$f(A|B,C) \propto f(A|B,C)\, f(B|C)$$

where the constant is hidden in the proportional sign $\propto$.

Application

In estimation we have

- A is the output y_t,
- B are parameters Θ and
- Cold data $d(t-1)$ (or $\{u_t,\, d(t-1)\}$).

In this way, the Bayes rule reads

$$f(\Theta|d(t)) = \frac{f(y_t|\psi_t,\Theta)\, f(\Theta|d(t-1))}{f(y_t|d(t-1))}$$

Remarks

1. *For model it holds* $f(y_t|u_t,d(t-1),\Theta) = f(y_t|\psi_t,\Theta)$.
2. *The natural conditions* $f(\Theta|u_t,d(t-1)) = f(\Theta|d(t-1))$ *are applied.*

5.5 Categorical distribution

The probability function of categorical distribution is

$$\begin{array}{c|cccc} y & 1 & 2 & \cdots & n_l \\ \hline f(y) & p_1 & p_2 & \cdots & p_{n_l} \end{array},$$

where p_i are probabilities, $p_i \geq 0$, $i = 1, 2, \cdots, n_l$ a $\sum_{i=1}^{n_l} p_i = 1$.

Alternative form for the pf is

$$f(y) = p_y, \ y = 1, 2, \cdots, n_l.$$

Model of discrete system is

$$f(y|\psi, \Theta) = \Theta_{y|\psi}.$$

and it can be expressed in the form of table (for $y \in \{1, 2\}$ and $\psi = [u, v]'$, where $u, v \in \{1, 2\}$

$$f(y|u, v)$$

$[u, v]$	$y = 1$	$y = 2$
1, 1	$\Theta_{1\|11}$	$\Theta_{2\|11}$
1, 2	$\Theta_{1\|12}$	$\Theta_{2\|12}$
2, 1	$\Theta_{1\|21}$	$\Theta_{2\|21}$
2, 2	$\Theta_{1\|22}$	$\Theta_{2\|22}$

,

$\Theta_{i|jk}$ are conditional probabilities $\Theta_{i|jk} \geq 0$, $\forall i, j, k$, $\sum_{i=1}^{2} \Theta_{i|jk} = 1$, $\forall j, k$.

For the purpose of estimation it is useful to express the model in so called **product form**

$$f(y|\psi, \Theta) = \prod_{i \in y^*} \prod_{\varphi \in \psi^*} \Theta_{i|\varphi}^{\delta(i|\varphi, y|\psi)}, \tag{5.3}$$

where i is index, φ is multiindex (vector index), y^*, ψ^* domains of variables and $\delta(i|\varphi, y|\psi)$ is Dirac function, i.e. it is one for $i|\varphi = y|\psi$ and zero otherwise.

5.6 Dirichlet distribution

A suitable distribution of model parameters in the case when model is categorical, is the Dirichlet one.

$$f(\Theta|d(t)) = \frac{1}{B(\nu_t)} \prod_{i \in y^*} \prod_{\varphi \in \psi^*} \Theta_{i|\varphi}^{\nu_{i|\varphi;t}}, \tag{5.4}$$

Here

ν_t is the statistics (with the same structure as the model has) ,

$B(\nu)$ is a multivariate beta function

$$B(\nu) = \prod_{\varphi \in \psi^*} \frac{\prod_{i \in y^*} \Gamma(\nu_{i|\varphi})}{\Gamma\left(\sum_{i \in y^*} \nu_{i|\varphi}\right)}, \tag{5.5}$$

where $\Gamma(\cdot)$ is gamma function defined by the formula

$$\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t)\, dt, \tag{5.6}$$

for which it holds

$$\Gamma(x+1) = x\Gamma(x), \ x \in R^+. \tag{5.7}$$

5.7 Normal distribution

We have normal regression model with regression vector ψ_t, regression coefficients θ and noise variance r, We denote $\Theta = \{\theta, r\}$. The model equation is

$$y_t = \psi_t' \theta + e_t, \ \ e_t \sim N(0, r).$$

The conditional pdf of the model is

$$f(y_t|\psi_t, \Theta) = \frac{1}{\sqrt{2\pi}} r^{-0.5} \exp\left\{-\frac{1}{2r}\left(y_t - \psi_t'\theta\right)^2\right\}. \tag{5.8}$$

Expectation

$$E[y_t|\psi_t, \Theta] = \psi_t'\theta,$$

variance

$$D[y_t|\psi_t, \Theta] = r.$$

For the purpose of estimation it is advantageous to modify the model in the following way:

- exponent is divided as follows

  $$y_t - \psi_t'\theta = -[-1\ \theta'] \begin{bmatrix} y_t \\ \psi_t \end{bmatrix} = -[y_t\ \psi_t] \begin{bmatrix} -1 \\ \theta \end{bmatrix}$$

  (the sign minus is formal).

- the square in the exponent is written as row times column

  $$\left(y_t - \psi_t'\theta\right)^2 = \left(y_t - \psi_t'\theta\right)\left(y_t - \psi_t'\theta\right) =$$
  $$= [-1\ \theta'] \begin{bmatrix} y_t \\ \psi_t \end{bmatrix} [y_t\ \psi_t] \begin{bmatrix} -1 \\ \theta \end{bmatrix} = [-1\ \theta']\, D_t \begin{bmatrix} -1 \\ \theta \end{bmatrix},$$

  where $D_t = \begin{bmatrix} y_t \\ \psi_t \end{bmatrix} [y_t\ \psi_t]$ is so called data matrix.

Model (5.8) now has the form

$$f(y_t|\psi_t, \Theta) = \frac{1}{\sqrt{2\pi}} r^{-0.5} \exp\left\{-\frac{1}{2r}[-1\ \theta']\, D_t \begin{bmatrix} -1 \\ \theta \end{bmatrix}\right\}. \tag{5.9}$$

5.8 Inverse Gauss-Wishart distribution

Its abbreviation is GiW

The distribution has the form

$$f\left(\Theta|d\left(t\right)\right) \propto r^{-0.5\kappa_t} \exp\left\{-\frac{1}{2r}\left[-1\,\theta'\right] V_t \begin{bmatrix} -1 \\ \theta \end{bmatrix}\right\}, \tag{5.10}$$

where κ_t and V_t are statistics (κ_t is the counter, V_t is the information matrix).

Matrix V_t is symmetric and positive definite and for computation of parameter point estimates it can be decomposed to sub-matrices

$$V_t = \begin{bmatrix} V_y & V'_{y\psi} \\ V_{y\psi} & V_\psi \end{bmatrix}, \tag{5.11}$$

where (for y_t scalar) V_y is a number, $V_{y\psi}$ is a column vector and V_ψ is a rectangle matrix.

5.9 Point estimate with quadratic criterion

The optimal point estimates must minimize the posted criterion. Here it is quadratic one

$$\min_{\hat{\Theta}_t} E\left[\left(\Theta - \hat{\Theta}_t\right)^2 |d\left(t\right)\right]. \tag{5.12}$$

We perform the square and than apply the expectation. Then we are going to use completion to square in $\hat{\Theta}$

$$\min_{\hat{\Theta}_t} E\left[\Theta^2 - 2\hat{\Theta}_t\Theta + \hat{\Theta}_t^2|d\left(t\right)\right] =$$

$$= \min_{\hat{\Theta}_t}\left\{E\left[\Theta^2|d\left(t\right)\right] - 2\hat{\Theta}_t E\left[\Theta|d\left(t\right)\right] + \hat{\Theta}_t^2\right\} = *1*$$

$\hat{\Theta}_t$ is a deterministic number

$$*1* = \min_{\hat{\Theta}_t}\left\{E\left[\Theta^2|d\left(t\right)\right] - E\left[\Theta|d\left(t\right)\right]^2 + E\left[\Theta|d\left(t\right)\right]^2 - 2\hat{\Theta}_t E\left[\Theta|d\left(t\right)\right] + \hat{\Theta}_t^2\right\} = *2*$$

we used the formula $D\left[\Theta\right] = E\left[\Theta^2\right] - E\left[\Theta\right]^2$ valid for the variance

$$*2* = \min_{\hat{\Theta}_t}\left\{D\left[\Theta|d\left(t\right)\right] + \left(\hat{\Theta}_t - E\left[\Theta|d\left(t\right)\right]\right)^2\right\} = D\left[\Theta|d\left(t\right)\right]$$

the minimum is

$$\hat{\Theta}_t = E\left[\Theta|d\left(t\right)\right]$$

as $D\left[\Theta|d\left(t\right)\right]$ is a constant with respect to $\hat{\Theta}_t$.

5.10 Point estimates of regression model parameters

MAP (Maximum Aposteriori Probability) estimation for normal regression model practically corresponds to minimization of quadratic criterion.

We look for maximum posterior pdf (which is a result of Bayesian estimation) (5.10)

$$f\left(\Theta|d\left(t\right)\right) \propto r^{-0.5\kappa} \exp\left\{-\frac{1}{2r}\left[-1\,\theta'\right] V \begin{bmatrix} -1 \\ \theta \end{bmatrix}\right\} =$$

$$= r^{-0.5\kappa} \exp\left\{-\frac{1}{2r}\left(V_y - 2\theta' V_{y\psi} + \theta' V_\psi \theta\right)\right\},$$

where we used the division of information vector V according to (5.11).

First we age going to estimate θ, i.e. to differentiate with respect to θ and lay the result equal to zero. It is a derivation of vectors according to vectors.

$$\frac{\partial f\left(\{\theta, r\}\,|d\left(t\right), r\right)}{\partial \theta} \propto r^{-0.5\kappa} \exp\left\{-\frac{1}{2r}\left[-1\,\theta'\right] V \begin{bmatrix} -1 \\ \theta \end{bmatrix}\right\} \left(\frac{-1}{2r}\right) \left(-2V_{y\psi} + 2V_\psi \theta\right) = 0.$$

From it he get

$$\hat{\theta} = V_\psi^{-1} V_{y\psi}. \tag{5.13}$$

We substitute the result into the posterior pdf and we obtain

$$\Lambda = V_y - 2\hat{\theta}' V_{y\psi} + \hat{\theta}' V_\psi \hat{\theta} =$$

$$= V_y - 2V_{y\psi}^{'} V_\psi^{-1} V_{y\psi} + V_{y\psi}^{'} V_\psi^{-1} V_\psi V_\psi^{-1} V_{y\psi},$$

and

$$\Lambda = V_y - V_{y\psi}^{'} V_\psi^{-1} V_{y\psi}. \tag{5.14}$$

The posterior pdf with the optimal point estimate of regression coefficient is(5.13)

$$f\left(r|d\left(t\right)\right) \propto r^{-0.5\kappa} \exp\left\{-\frac{\Lambda}{2r}\right\}.$$

We differentiate it and lay equal to zero

$$-\kappa \frac{1}{2r} + \Lambda \frac{1}{r^2} = 0,$$

From it we have

$$\hat{r} = \frac{\Lambda}{\kappa}. \tag{5.15}$$

$\hat{\theta}$ a $\hat{r}$ are point estimates which we are seeking for.

5.11 Point estimates of categorical model parameters

Here, the point estimates of parameters are given by a mere normalization of rows of or the statistics matrix ν_t

$$\hat{\Theta}_{y|\psi;t} = \frac{\nu_{y|\psi;t}}{\sum_{i\in y^*}\nu_{i|\psi;t}}, \ \forall y \in y^* \,\mathrm{a}\, \psi \in \psi^*. \tag{5.16}$$

The point estimate is an expectation of parameter with posterior pdf (5.4) - for lucidity we skip the time index t

$$\hat{\Theta}_{y|\psi} = E\left[\Theta_{y|\psi}|d\left(t\right)\right] = \int_0^\infty \Theta_{y|\psi} f\left(\Theta|d\left(t\right)\right) d\Theta =$$

$$= \frac{1}{B\left(\nu\right)} \int_0^\infty \Theta_{y|\psi} \prod_{i\in y^*}\prod_{\varphi\in\psi^*} \Theta_{i|\varphi}^{\nu_{i|\varphi}} d\Theta = *1*,$$

where beta function B is given in (5.5). Formally we express the model in a product form (5.3)

$$\Theta_{y|\psi} = \prod_{i\in y^*}\prod_{\varphi\in\psi^*} \Theta_{i|\varphi}^{\delta(i|\varphi,y|\psi)}$$

and substitute. We continue

$$*1* = \frac{1}{B\left(\nu_t\right)} \int_0^\infty \prod_{i\in y^*}\prod_{\varphi\in\psi^*} \Theta_{i|\varphi}^{\nu_{i|\varphi}+\delta(i|\varphi,y|\psi)} d\Theta =$$

$$= \frac{1}{\prod_{\varphi\in\psi^*} B\left(\nu_\varphi\right)} \prod_{\varphi\in\psi^*} \int_0^\infty \prod_{i\in y^*} \Theta_{i|\varphi}^{\nu_{i|\varphi}+\delta(i|\varphi,y|\psi)} d\Theta_{y|\psi} = *2*,$$

where

$B\left(\nu_\varphi\right) = \frac{\prod_{i\in y^*}\Gamma\left(\nu_{i|\varphi}\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\varphi}\right)}$ according to (5.5)

we use the assumption of independence between parameters from different components.

For individual components it holds

$$\int_0^\infty \prod_{i\in y^*} \Theta_{i|\varphi}^{\nu_{i|\varphi}+\delta(i|\varphi,y|\psi)} d\Theta_{y|\psi} = \begin{cases} B\left(\nu_\varphi\right) & \text{pro } \delta = 0, \\ B\left(\nu_\psi + 1\right) & \text{pro } \delta = 1. \end{cases}$$

The terms with $\delta = 0$ are canceled

$$*2* = \frac{B\left(\nu_\psi + \delta\left(i,y\right)\right)}{B\left(\nu_\psi\right)} = \frac{\frac{\prod_{i\in y^*}\Gamma\left(\nu_{i|\psi}+\delta(i,y)\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}+1\right)}}{\frac{\prod_{i\in y^*}\Gamma\left(\nu_{i|\psi}\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}\right)}} = *3*\,.$$

and again the terms for which $y \neq i$ are canceled, too, and we get

$$*3* = \frac{\frac{\Gamma\left(\nu_{y|\psi}+1\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}+1\right)}}{\frac{\Gamma\left(\nu_{y|\psi}\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}\right)}} = \frac{\frac{\nu_{y|\psi}}{\sum_{i\in y^*}\nu_{i|\psi}}\frac{\Gamma\left(\nu_{y|\psi}\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}\right)}}{\frac{\Gamma\left(\nu_{y|\psi}\right)}{\Gamma\left(\sum_{i\in y^*}\nu_{i|\psi}\right)}} = \frac{\nu_{y|\psi}}{\sum_{i\in y^*}\nu_{i|\psi}}.$$

In the above derivation we also have used the properties of the gamma function (5.7).

This completes the proof of (5.16).

5.12 Logistic regression in details

Derivative of likelihood for logistic regression

Derivative of logarithm for likelihood $\ln L$ with the model(**??**) with respect to Θ is

$$\frac{\partial}{\partial\Theta}\ln L\left(\Theta\right) = \sum_{\tau=1}^{t}\left[y_\tau\psi_\tau - \frac{\exp\left(z_\tau\right)}{1+\exp\left(z_\tau\right)}\psi_\tau\right] = \sum_{\tau=1}^{t}\left(y_\tau - p_\tau\right)\psi_\tau,$$

where according to (**??**) $z_\tau = \psi_\tau\Theta$ a and so$dz_\tau/d\Theta = \psi_\tau$. Further we denote

$$p_\tau = \frac{\exp\left(z_\tau\right)}{1+\exp\left(z_\tau\right)} = P\left(y_t = 1|\psi_\tau,\Theta\right).$$

The second derivativeln L with respect to Θ is

$$\frac{\partial^2}{\partial\Theta^2}\ln L\left(\Theta\right) = \frac{\partial}{\partial\Theta}\sum_{\tau=1}^{t}\left(y_\tau - p_\tau\right)\psi_\tau = \sum_{\tau=1}^{t}\frac{\partial}{\partial\Theta}p_\tau\psi_\tau = \sum_{\tau=1}^{t}p_\tau\left(1-p_\tau\right)\psi_\tau'\psi_\tau,$$

as

$$\frac{\partial}{\partial\Theta}p_\tau = \frac{\partial}{\partial\Theta}\frac{\exp\left(z_\tau\right)}{1+\exp\left(z_\tau\right)} = \frac{\exp\left(z_\tau\right)\psi_\tau'\left(1+\exp\left(z_\tau\right)\right) - \exp\left(z_\tau\right)\exp\left(z_\tau\right)\psi_\tau'}{\left(1+\exp\left(z_\tau\right)\right)^2} =$$

$$= \frac{\exp\left(z_\tau\right)\psi_\tau'}{\left(1+\exp\left(z_\tau\right)\right)^2} = \left(\frac{\exp\left(z_\tau\right)}{1+\exp\left(z_\tau\right)}\frac{1}{1+\exp\left(z_\tau\right)}\right)\psi_\tau' = p_\tau\left(1-p_\tau\right)\psi_\tau'.$$

For numerical maximization it is advantageous to use Newton algorithm (both the derivatives are analytical).

Newton algorithm

Let us denote $g\left(x\right)$ the function to be minimized; here $x = \left[x_1, x_2\cdots x_n\right]'$. The gradient g' and Hess matrix g'' are

$$g'\left(x\right) = \begin{bmatrix} \frac{\partial g}{\partial x_1} \\ \frac{\partial g}{\partial x_2} \\ \cdots \\ \frac{\partial g}{\partial x_n} \end{bmatrix}$$

$$g''(x) = \begin{bmatrix} \frac{\partial^2 g}{\partial x_1^2} & \frac{\partial^2 g}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n} \\ \frac{\partial^2 g}{\partial x_2 \partial x_1} & \frac{\partial^2 g}{\partial x_2^2} & \cdots & \frac{\partial^2 g}{\partial x_2 \partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 g}{\partial x_n \partial x_1} & \frac{\partial^2 g}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_n^2} \end{bmatrix}.$$

The algorithm starts at the point $x^{(0)}$ and generates further points $x^{(1)}, x^{(2)}, \cdots$ as follows: We take Taylor expansion of g at $x^{(i)}$ and use its first three terms (quadratic function)

$$g(x) \doteq g\left(x^{(i)}\right) + g'\left(x^{(i)}\right)\left(x - x^{(i)}\right) + \frac{1}{2}g''\left(x^{(i)}\right)\left(x - x^{(i)}\right)^2.$$

For the next point $x^{(i+1)}$ we minimize this quadratic function

$$g'\left(x^{(i)}\right) + g''\left(x^{(i)}\right)\left(x^{(i+1)} - x^{(i)}\right) = 0$$

from which we have

$$x^{(i+1)} = x^{(i)} - \frac{g'\left(x^{(i)}\right)}{g''\left(x^{(i)}\right)}.$$

We repeat it till the estimates stabilize.