



národní
úložiště
šedé
literatury

IINC Software

Jiřina, Marcel
2015

Dostupný z <http://www.nusl.cz/ntk/nusl-200884>

Dílo je chráněno podle autorského zákona č. 121/2000 Sb.

Tento dokument byl stažen z Národního úložiště šedé literatury (NUŠL).

Datum stažení: 04.05.2024

Další dokumenty můžete najít prostřednictvím vyhledávacího rozhraní [nusl.cz](http://www.nusl.cz) .



Institute of Computer Science
Academy of Sciences of the Czech Republic

IINC Software

Marcel Jiřina

Technical Report No. V-1225

October 2015

Abstract

In this report we present the source code in c++ of the IINC family of classifiers and k-NN classifiers with possibility to choose one from fifteen different metrics including L1, L2, Mahalanobis and Hassanat metrics. One can also use weighting of samples of the learning data. A short help of this program is described. The source code is written so that it can be compiled and run from the command line under Windows as well as under Linux environment.

Keywords:

Multidimensional data, Classifier, Distance, Metrics, Hassanat metrics, k-NN, IINC.

Acknowledgement

This work was supported by the Czech Ministry of Education, Youth and Sports in the project No. LG15047 "Cooperation on Experiments in the Fermi National Laboratory, USA" of the programme INGO II.

Contents

Introduction	3
How to use the program	3
References	5
Main program listing	6
Relatively universal functions	34

Introduction

This program contains algorithms k-NN methods and algorithms of the family of scaling-based classifiers. This family we call “IINC” and consist of three main methods:

- The “local” method that uses a local value of the scaling exponent (the distribution mapping exponent) [1]
- The “global” method that uses a global value of scaling exponent, the correlation dimension [2]
- The IINC (Inverted Indices of Neighbors Classifier). It uses inverted ranks of neighbors $1/i$ instead of the $1/r^q$, where q is the scaling exponent in two previously mentioned methods [3].

For a short description of the core of these methods see [4].

All methods can use one of fifteen metrics among them the L_p metrics with any positive p . (Of course, if $p < 1$ it is not a metric.) Also the Mahalanobis metric is implemented and an advanced algorithm is used [6].

There are many features from selecting a method, metric, what kind of regression will be used for stating the scaling exponent and so on. One can also use sample weights either included in data file or in a separate file. In the IINC family of algorithms samples weights are applied so that ratio $1/r^q$ or $1/i$ is multiplied by the sample weight. In k-NN algorithms we divide the distance by the sample weight to make it larger and less influential when weight is small. Feature weights are not considered.

In k-NN methods also suppression of the influence of the crossing phenomenon can be applied. It can lead to better results when k larger than five is used.

To get a metric space from the data space a normalization of each feature to zero mean and unit standard deviation is used. This normalization is applied even when metric is defined so that such normalization is not needed.

The program can be easily compiled and then run from a command line under Windows as well as under Linux environment. Input data and output files are all in the text format. Simplified IINC program exists also in MS Excel, see [5].

How to use the program

Here a short help is listed. This help appears when the runtime (exe) module is run without parameters. Text in brackets [] means optional parameter or parameters.

```
IINC ver.10/2015.
```

```
with sample weights and with crossing for k-NN methods
```

```
Usage: ./IINC.exe LearningFile TestFile [named parameters]
```

```
LearningFile: each row: inputs ClassMark [sample weight]
```

```
No other data is supposed on the line.
```

```
Max. 100 classes are supposed.
```

```
Task dimension DIM and the LearningFile size are limited by the size of  
allocatable memory only.
```

```
TestFile: each row: inputs [ClassMark]
```

```
Only DIM or DIM+1 items are read from the line.
```

```
Named parameters: Form: ParName=Parvalue
```

```
FileType: a type of NN and Test files. Default=0
```

```
=0 Reading lines as numbers including ClassMark.
```

```
=1 Reading NOOfInputs numbers and Class string.
```

```
=2 Reading pairs (index of word, word count) until (-1 -1) then Class string.
```

```
In this case state the dimension using named parameter DIM=...
```

```
SWEI: the name of the file of sample weights, eventually including the path.
```

```
if SWEI=1 then the file of sample weights LearningFile.wei is searched.
```

```
Default SWEI=0, i.e. no sample weights are present.
```

```
Weights should be positive only; else absolute values are used.
```

The file of weights must have the same number of rows as the learning file
If the file of weights does not have the same number of rows as
the LearninfFile error message appears and program stops.
sweic: the column number in the file of smoles weights.
default 1. If the item does not exist the weight is set to 1.
DIM: the task dimension when FileType 2 is used else it is stated automatically
according to the number of items on the first row of the LearningFile
DME: selection of the method. Default 0.
if DME=-1 nearest neighbor methods. (Eventually weighted: distance divided by
weight.)
The ize of the neighborhood is given by parameter SEL.
if DME=0 IINC (1/i) method (no influence of SEL or REG). Eventually weighted.
if DME=1 QCregre method that uses an additive constant not DME (not weighted).
if DME=11 QCregre method with indexing over the whole learn.set.
if DME=2 DME-local method (SFSloc7 for SEL=0). Uses DME slope. Eventually
weighted.
if DME=3 CDglobal: uses correlation dimension globally. Use AdHoc
as the number of DMEs used for CD estimation; default 100. Eventually
weighted.
if DME=4 CIntegral method; use SEL>=6 else neighborhood is 5.
Uses local comparison of CI distribution functions (not weighted).
else default DME=0 is used.
SEL: selection strategy for neighborhood. Default=2.
if SEL<0 use abs(SEL) nearest points.
if SEL=0 use the whole learning set and q=2*q.
if SEL=2 use one half of samples of the learn.set.
if SEL=3 use first sqrt(No. of samples of the learn.set).
if SEL=4 as SEL=2 starting from 1/3 of that.
if SEL=5 as SEL=3 starting from 1/3 of that.
if SEL>=6 use SEL nearest points.
else default SEL=2 is used.

REG: for DME>0 use a regression as follows.
if REG=0 standard linear regression - default.
if REG=1 weighted p/ln(p) linear regression.
if REG=2 hyperbolic regression. NO weighting, NO robust.
if REG=3 Fabian beta-prime weighted linear LMS regression.
if REG=4 robust by deletion of large residuals.
if REG=5 Takens CD/DME estimator (computes C too) [with ad hoc coeff. (0.8)].

Metrics; default: Metrics=2.
if Metrics: >= 1; Lp metrics.
if Metrics: = -1; Mahalanobis metrics derived from the whole learning set)
if Metrics: = -2; Mahalanobis class dependent metrics.
if Metrics: = -3; Orloci metrics.
if Metrics: = -4; Angular semimetrics.
if Metrics: = -5; Clark metrics.
if Metrics: = -6; Lorentz metrics.
if Metrics: = -7; Canberra metrics.
if Metrics: = -8; Bray-Curtis (nearly) metrics.
if Metrics: = -9; Intersection (nearly) metrics.
if Metrics: =-10; Cayley-Klein-Hilbert metrics.
if Metrics: =-11; Weierstrass metrics.
if Metrics: =-12; Hassanat metrics.
else Euclidean metrics.

Cross; default: Cross=0.
if Cross = 0; Do not test the crossing point.
if Cross > 0; Use the Cross percent from the crossing point.
(Cross lies between 0 and 100 percent.
(Eventually, the crossing includes weights.)

AdHoc: a constant for fine tuning in some cases. Default 1.

File of results: filename = TestFile.res:
The first row: lrn file name, tst file name, No. of inputs, ValueOfClass1
Other rows each:
Input values (their number according to No. of inputs),
class if given else 0, class found, probabilities of classes, sample No.

Additional files generated:

register.txt:
One row is appended for each task and consists of task description,
number of testing samples and no. of errors.
howfar.txt:
Shows percentage of samples already processed.
(good for very large data sets.)
Hit ENTER to continue

References

- [1] Jiřina, Marcel - Jiřina jr., M.: Utilization of Singularity Exponent in Nearest Neighbor Based Classifier. Journal of Classification. Vol. 30, No. 1 (2013), pp. 3-29. ISSN 0176-4268 <http://hdl.handle.net/11104/0208364> .
- [2] Marcel Jiřina, Marcel Jiřina, jr. (2014) Correlation dimension based Classifier. IEEE Transactions on Cybernetics, Vol. 44, No. 12 , pp. 2253-2263. ISSN 2168-2267
- [3] Jiřina, Marcel - Jiřina jr., M.: Classification Using Zipfian Kernel. Journal of Classification (Springer), Vol. 32, 12 April 2015, pp. 305-326. ISSN 0176-4268. Electronic version available at http://www.library.sk/ar/cav/cs/contapp/?idx=cav_un_epca*0420984&repo=crepo1&key=84163352979
- [4] Marcel Jirina: Exponentially scaled point processes and data classification. (Invited paper) Proceedings of the 2014 International Conference on Pure Mathematics, applied Mathematics, Computational methods (PMAMCM2014), Santorini Island, Greece, July 17-21, 2014, pp. 179-186 (Paper MATH-27.pdf).
- [5] Marcel Jiřina: IINC classifier for MS Excel. The principle, method and Program. Technical Report No. V-1199, Institute of Computer Science, Academy of Sciences of the Czech Republic, October 2014, 9 pp.
- [6] M. Benzi and J.K. Cullum and M. Tůma. Robust Approximate Inverse Preconditioning for the Conjugate Gradient Method, SIAM J. Sci. Comput. 22 (2000), 1318--1332.

Main program listing

```
/*Changes 08/2015>
    Crossing included if desired; variable long Cross
/*Changes 02/2014:
    Mahalanobis distance included
    Changed way of method and its parameters control;
        DME and SEL parameters slightly modified,
        WEI parameter discarded.
    Corrected error appearing when test data without class were used as testing
    set.
    Distances are now computed by universal separate function and corresponding
    places call this function.

*/
#include    <math.h>
#include    <stdlib.h>
#include    <stdio.h>
#include    <string.h>
#include    "MyFu.h"

#define    real double    //float //double a v fscanff zmenit %g na %lg
//#define    int long
//#define    and &&        //define    or ||        //define    not !

void CovarMatrix(double LearnArr[], long Dimension, long LearnSamples, long Class,
double CovMat[], long Count[]) {
    //Computes the covariance matrix CovMat for Class or for all samples if
    Class<0.
    //zero means are supposed !!
    long i,j,k,kk,trida;
    double x;
    if(Class<0) {
        x=1/(double)LearnSamples;
        kk=0;
    }
    else {
        x=1/(double)Count[Class];
        kk=Class;
    }
    //zeros to kk-th layer of CovMat
    for(i=0;i<Dimension;i++)
        for(j=0;j<Dimension;j++)
            CovMat[kk*Dimension*Dimension+i*Dimension+j]=0;

    //fill-in the CovMat (the class corresponding layer)
    for(k=0;k<LearnSamples;k++){
        trida=(long)LearnArr[k*(Dimension+1)+Dimension];
        if(trida==Class || Class<0) {
            for(i=0;i<Dimension;i++)
                for(j=0;j<Dimension;j++){
                    CovMat[kk*Dimension*Dimension+i*Dimension+j]
+=LearnArr[k*(Dimension+1)+i]*LearnArr[k*(Dimension+1)+j]*x;
                }
            }
    }
} // end of void CovarMatrix.

double MahalanobisDist(double x[], double y[], double ZCovMat[], long Dimension,
long Class){
    //computes Mahalanobis dostance between x and y using CovMat already known
    //via solution of system of linear equations
    //zero means are supposed !!
    long i, j, kk;
    double xx;
    double* ZZ= new double [Dimension*Dimension]; // working ZCovMat, i.e.
```

```

        if(Class<0) kk=0; else kk=Class;
        for(i=0;i<Dimension;i++){
            for(j=0;j<Dimension;j++){
                ZZ[i*Dimension+j]=ZCovMat[kk*Dimension*Dimension+i*Dimension+j];
//                printf("%10lg ", CM[i*Dimension+j]);
            }
//            printf("\n");
        }
//        getchar();
//double mahala(double u[], double v[], double z[], long n){
xx=mahala(x, y, ZZ, Dimension);
delete ZZ; // <<<<NUTNE!!!
return xx;
}

double Orloci(double x[], double y[], long Dimension){// = sqrt(2*(1-
<x,y>/(|x|*|y|))
    long i, j, kk;
    double xx,yy,xy;
    xx=0; yy=0; xy=0;
    for(i=0;i<Dimension;i++){
        xx+=Na2(x[i]);
        yy+=Na2(y[i]);
        xy+=x[i]*y[i];
    }
    xx=sqrt(xx);
    yy=sqrt(yy);
    return sqrt(2*(1-xy/(xx*yy)));
}

double Angular(double x[], double y[], long Dimension){//=
arccos(<x,y>/(|x|*|y|))
    long i, j, kk;
    double xx,yy,xy;
    xx=0; yy=0; xy=0;
    for(i=0;i<Dimension;i++){
        xx+=Na2(x[i]);
        yy+=Na2(y[i]);
        xy+=x[i]*y[i];
    }
    xx=sqrt(xx);
    yy=sqrt(yy);
    return acos(xy/(xx*yy));
}

double Clark(double x[], double y[], long Dimension){//Clark metrics =
sqrt{1/nSUM[(xi-yi)/(|xi|+|yi|)]^2}
    long i, j, kk;
    double xx,yy,xy;
    xx=0; yy=0; xy=0;
    for(i=0;i<Dimension;i++) xx+=Na2((x[i]-y[i])/(abs(x[i])+abs(y[i]))));
    return sqrt(xx/Dimension);
}

double Lorentz(double x[], double y[], long Dimension){//Lorentz metrics =
SUM[ln(1+|xi-yi|)]
    long i, j, kk;
    double xx,yy,xy;
    xx=0; yy=0; xy=0;
    for(i=0;i<Dimension;i++) xx+=log(1+abs(x[i]-y[i]));
    return xx;
}

double Canberra(double x[], double y[], long Dimension){//Canberra metrics =
SUM[|xi-yi|/(|xi|+|yi|)]
    long i, j, kk;

```



```

double xx,yy,xy;
xx=0; yy=0; xy=0;
for(i=0;i<Dimension;i++) {
    xy=abs(x[i])+abs(y[i]);
    if(xy>1.0e-10) xx+=abs(x[i]-y[i])/(xy);
}
return xx;
}

double BrayCurtis(double x[], double y[], long Dimension){//Bray-Curtis (nearly)
metrics = SUM|xi-yi|/SUM(xi+yi)
long i, j, kk;
double xx,yy,xy;
xx=0; yy=0; xy=0;
for(i=0;i<Dimension;i++) {
    xx+=abs(x[i]-y[i]);
    yy+=x[i]+y[i];
}
return xx/yy;
}

double Intersection(double x[], double y[], long Dimension){//Intersection (nearly)
metrics = 1-SUM[min(xi,yi)]/min[sum xi,sum yi]
long i, j, kk;
double xx,yy,xy;
xx=0; yy=0; xy=0;
for(i=0;i<Dimension;i++) {
    xy+=x[i]<y[i]?x[i]:y[i];
    xx+=x[i];
    yy+=y[i];
}
return 1-xy/(xx<yy?xx:yy);
}

double CayleyKleinHilbert(double x[], double y[], long Dimension){
//Cayley-Klein-Hilbert metrics = arccosh{abs[1-<x,y>]/[sqrt abs(1-<x,x>)*sqrt
abs(1-<y,y>)]
long i, j, kk;
double xx,yy,xy;
xx=0; yy=0; xy=0;
for(i=0;i<Dimension;i++) {
    xy+=x[i]*y[i];
    xx+=Na2(x[i]);
    yy+=Na2(y[i]);
}
return 0;//acosh(abs(1-xy)/(sqrt(abs(1-xx))*sqrt(abs(1-yy))));
}

double Weierstrass(double x[], double y[], long Dimension){
//Weierstrass metrics = arccosh{abs[sqrt(1+<x,x>)*sqrt(1+<y,y>)-<x,y>]}
long i, j, kk;
double xx,yy,xy;
xx=0; yy=0; xy=0;
for(i=0;i<Dimension;i++) {
    xy+=x[i]*y[i];
    xx+=Na2(x[i]);
    yy+=Na2(y[i]);
}
return 0;//acosh(abs(sqrt(1+xx)*sqrt(1+yy)-xy));
}

double Hassanat(double x[], double y[], long Dimension){
// Hassanat metrics =

```

```

// = sum{if min xi,yi>0 then 1-(1+min xi,yi)/(1+max xi,yi) else 1-(1+min
xi,yi + abs min xi,yi)/(1+max xi, yi +abs min xi,i)}
long i, j, kk;
double dist,minx,maxx;
dist=0;
for(i=0;i<Dimension;i++) {
    minx=x[i]<y[i]?x[i]:y[i];
    maxx=x[i]>y[i]?x[i]:y[i];
    dist+=minx>=0?1-(1+minx)/(1+maxx):1-
(1+minx+fabs(minx))/(1+maxx+fabs(maxx));
}
return dist;
}

```

```

double distance(double x[], double y[], double ZCovMat[], long Dimension,
double Metrics, long Class) {
double dist2;
long j;
if(Metrics>0){//then Lp metrixs
    dist2=0;
    for(j=0;j<Dimension;j++) dist2+=Na(x[j]-y[j], Metrics);
    return Na(dist2, 1/Metrics);
}
if((long)(Metrics-0.001)==-1){//then Mahalanobis independent of class
    return MahalanobisDist(x, y, ZCovMat, Dimension, -1);
}
if((long)(Metrics-0.001)==-2){//then Mahalanobis dependent of class CDM
    return MahalanobisDist(x, y, ZCovMat, Dimension, Class);
}
if((long)(Metrics-0.001)== -3){// Orloci metrics = sqrt(2*(1-
<x,y>/(|x|*|y|)) \n");
    return Orloci(x, y, Dimension);
}
if((long)(Metrics-0.001)== -4){// Angular semimetrics =
arccos(<x,y>/(|x|*|y|)) \n");
    return Angular(x, y, Dimension);
}
if((long)(Metrics-0.001)== -5){// Clark metrics = sqrt{1/nSUM[(xi-
yi)/(|xi|+|yi|)]^2} \n");
    return Clark(x, y, Dimension);
}
if((long)(Metrics-0.001)== -6){// Lorentz metrics = SUM[ln(1+|xi-yi|)]
\n");
    return Lorentz(x, y, Dimension);
}
if((long)(Metrics-0.001)== -7){// Canberra metrics = SUM[|xi-
yi|/(|xi|+|yi|)] \n");
    return Canberra(x, y, Dimension);
}
if((long)(Metrics-0.001)== -8){// Bray-Curtis (nearly) metrics = SUM|xi-
yi|/SUM(xi+yi) \n");
    return BrayCurtis(x, y, Dimension);
}
if((long)(Metrics-0.001)== -9){// Intersection (nearly) metrics = 1-
SUM[min(xi,yi)]/min[sum xi,sum yi] \n");
    return Intersection(x, y, Dimension);
}
if((long)(Metrics-0.001)==-10){// Cayley-Klein-Hilbert metrics =
arccosh{abs[1-<x,y>]/[sqrt abs(1-<x,x>)*sqrt abs(1-<y,y>)]} \n");
    return CayleyKleinHilbert(x, y, Dimension);
}
if((long)(Metrics-0.001)==-11){// Weierstrass metrics =
arccosh{abs[sqrt(1+<x,x>)*sqrt(1+<y,y>)-<x,y>]} \n");
    return Weierstrass(x, y, Dimension);
}
if((long)(Metrics-0.001)==-12){// Hassanat metrics =

```

```

        // = sum(if min xi,yi>0 then 1-(1+min xi,yi)/(1+max xi,yi) else 1-
(1+min xi,yi + abs min xi,yi)/(1+max xi, yi +abs min xi,i)}
        return Hassanat(x, y, Dimension);
    }
/**/
    //else Euclidean metrics
    dist2=0;
    for(j=0;j<Dimension;j++) dist2+=Na(x[j]-y[j], 2.0);
    return Na(dist2, 1/2.0);
}

void heurQC(const void *Q, const void *C, int Dim, TDist Dist[], long zacatek, long
konec){
    //heuristics for strange values of Q s opravou C počítáno prostě z prvního a
posledního (index) bodu.
    //called as heurQC(&QQ[2],&CC[6], Dimense, Dist, zac, kon);
    double q,c;
    q=(double*)Q;
    c=(double*)C;
    if(zacatek<1) zacatek=1;
    if(q<=0 || q>Dim && Dim<6 || q>Dim-1 && Dim>5){
        q=(log(konec)-log(zacatek))/(log(Dist[konec].D)-
log(Dist[zacatek].D)); //distribution mapping exponent
        c=log(konec)-q*log(Dist[konec].D); //regression constant (it estimates
distribution density in the query point)
    }
    //a kdyby i tohle vychazelo spatne, tak jako kdysi:
    if(q<=0) q=Dim;
    if (Dim<6) q=(q>Dim?Dim:q);
    else q=(q>Dim-1?Dim-1:q);
    *(double*)Q=q;
    *(double*)C=c;
}

void HypRegre2(TDist Dist[], double q[], double C[], double LearnArr[], long
LearnSamples, long Dimension,
                long zacatek, long konec, long Count[], long trida, long
whole, long residuals, long WEI){
    //supposed distances sorted in ascending order
    //if whole>0 then use indexing over the whole learning set, otherwise count
samples of class "trida" only.
    double ri, residu, maxresidu, lnIapr, sumresidu;
    double Sa, Sb, Sc, Sd, Se, Sf, Sr, Ss, St, lnpi, lnri;
    long i, j, index, pocet, indresidu;
    char tab;
    tab=9;

    /*otisk Dist
char tab; tab=9;
for(i=0;i<50;i++) printf
("%i%c%i%lg%c%i\n",i+1,tab,Dist[i].I,tab,Dist[i].D,tab,(long)(0.001+LearnArr[Dist
[i].I*(Dimension+1)+Dimension]));
getchar();/**/

    Sa=0;Sb=0;Sc=0;Sd=0;Se=0;Sf=0;Sr=0;Ss=0;St=0;
    index=0;
    for(j=0;j<LearnSamples && index<=konec;j++) {
        i=(long)(0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
        if(i==trida || whole>0){ // if whole>0 then take all points without
respect to class.
            index++; //points of the class trida are indexed in all cases one
after another 1, 2, ...
            if(index>zacatek && index<=konec && Dist[j].V>0) { //jen v danem
intervalu a nepasivovane body, ktere se ale pocitaji do "pocet"

```

```

(1, 2, ...)
//count the number of truly used points for regression

ri=Dist[j].D; // distance in a given Metrics
lnri=log(ri);
lnpi=log((double)index/(double)Count[trida]);
Sa++;
Sb=Sb-lnpi;
Sc=Sc-1/lnpi;//+
Sd=Sd+lnpi*lnpi;
Se=Se++;//--
Sf=Sf+1/(lnpi*lnpi);
Sr=Sr-lnri;
Ss=Ss+lnpi*lnri;
St=St-lnri/lnpi;//-
    }
}
} //tim mame udaje do regresni rovnice
q[trida]= (Sd*(Sa*Sf-Sc*Sc)+Se*(Sb*Sc-Sa*Se)+Sb*(Sc*Se-Sb*Sf))
//      /(Ss*(Sa*Sf-Sc*Sc)+St*(Sb*Sc-Sa*Se)+Sr*(Sc*Se-Sb*Sf)); // =D/DQ
C[trida]= (Ss*(Se*Sc-Sb*Sf)+St*(Sb*Se-Sd*Sc)+Sr*(Sd*Sf-Se*Se))
//      /(Ss*(Sa*Sf-Sc*Sc)+St*(Sb*Sc-Sa*Se)+Sr*(Sc*Se-
Sb*Sf)); // =DkQ/DQ=(k/q)*q
C[trida]=-log(fabs(C[trida]));
//a= (Ss*(Sb*Sc-Sa*Sc)+St*(Sa*Sd-Sb*Sb)+Sr*(Sb*Sc-Sc*Sc))
//    /(Sd*(Sa*Sf-Sc*Sc)+Se*(Sb*Sc-Sa*Se)+Sb*(Sc*Se-Sb*Sf)); // =Da/D

if(residuals>0) { //RESIDUALS: compute residuals int Dist[].der
maxresidu=0; indresidu=0; index=0; sumresidu=0;
for(j=0; j<LearnSamples && index<=konec; j++) {
    i=(long)(0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
    if(i==trida || whole>0){
        index++;
        if(index>zacatek && index<=konec && Dist[j].V>0) { //jen v
danem intervalu a nepasivovane body, ktere se ale pocitaji do "pocet"
            ri=Dist[j].D; // distance in a given Metrics
            lnIapr=C[trida]+q[trida]*log(ri); //approx in the ln
world
            residu=(double)log(index)-lnIapr; //residuuum in the
ln world

            Dist[j].der=residu; //lnIapr; //exp(lnIapr);
            sumresidu+=residu; //OK, truly, sum of residuals

practically equals to zero.

            if(maxresidu<fabs(residu)) {
                maxresidu=fabs(residu);
                indresidu=j;
            }
        }
        pocet++;
    }
} // nakonec mame indresidu pro pasivaci nejhorsiho bodu
Dist[indresidu].V=0; //a pasivaci hned provedeme
Dist[indresidu].der=0;
} //end of part RESIDUALS/**/
}

void LinRegre2(TDist Dist[], double q[], double C[], double LearnArr[], long
LearnSamples, long Dimension,
                long zacatek, long konec, long Count[], long trida, long
whole, long residuals,
                long WEI){
    //supposed distances sorted in ascending order
    //if whole>0 then use indexing over the whole learning set, otherwise count
samples of class "trida" only.
    double Slnri2, Slnri, Slni, Slnilnri, ri, residu, maxresidu, lnIapr,
sumresidu, logri, logindex, W;
    long i, j, index, pocet, indresidu;
    char tab;

```

```

tab=9;

/*otisk Dist
char tab; tab=9;
for(i=0;i<50;i++) printf
("%i%c%i%lg%c%i\n",i+1,tab,Dist[i].I,tab,Dist[i].D,tab,(long)(0.001+LearnArr[Dist
[i].I*(Dimension+1)+Dimension]));
getchar();/**/

Slnri2=0; Slnri =0; Slni =0; Slnilnri=0;
pocet=0; index=0;
for(j=0;j<LearnSamples && index<=konec;j++) {
    i=(long)(0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
    if(i==trida || whole>0){// if whole>0 then take all points without
respect to class.
        index++;//points of the class trida are indexed in one after
another 1, 2, ...
        if(index>zacatek && index<=konec && Dist[j].V>0) {//jen v danem
intervalu a nepasivovane body, ktore se ale pocitaji do "pocet"
            pocet++;//count the number of truly used points for
regression (1, 2, ...)
            ri=Dist[j].D; /* distance in a given Metrics*/
            if(WEI==1){ //..*index/log(index) =>
                W=index==1?1:index/log(index); // (index+1)/log(index+1);
                logri=W*log(ri);
                logindex=(double)index; //=W*log(index);
            }
            else {
                logri=log(ri);
                logindex=log(index);
            }
            Slnri2=Slnri2+logri*logri;
            Slnri =Slnri+logri;
            Slni =Slni+logindex;
            Slnilnri=Slnilnri+logindex*logri;
//printf("%9lg %9i %9lg %9lg %9lg %9lg\n", logri, index, logindex, Slnri2,
Slnri, Slni, Slnilnri);
        }
    }
}
} //thus we have data for regression equation; index is the last point used
q[trida]= (pocet*Slnilnri-Slnri*Slni)/(pocet*Slnri2-
Slnri*Slnri); //distribution mapping exponent
C[trida]=(Slni*Slnri2-Slnri*Slnilnri)/(pocet*Slnri2-Slnri*Slnri); //regression
constant (it estimates distribution density in the query point)
//getchar();

if(residuals>0) { //RESIDUALS: compute residuals int Dist[].der
    maxresidu=0; indresidu=0; index=0; sumresidu=0;
    for(j=0;j<LearnSamples && index<=konec;j++) {
        i=(long)(0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
        if(i==trida || whole>0){
            index++;
            if(index>zacatek && index<=konec && Dist[j].V>0) { //jen v
danem intervalu a nepasivovane body, ktore se ale pocitaji do "pocet"
                ri=Dist[j].D; /* distance in a given Metrics*/
                lnIapr=C[trida]+q[trida]*log(ri); //approx in the ln
world
                residu=(double)log(index)-lnIapr; //residuum in the
ln world
                Dist[j].der=residu; //lnIapr; //exp(lnIapr);
                sumresidu+=residu; //OK, truly, sum of residuals
practically equals to zero.
                if(maxresidu<fabs(residu)){
                    maxresidu=fabs(residu);
                    indresidu=j;
                }
            }
        }
    }
}

```



```

//getchar();

if(residuals>0) { //RESIDUALS: compute residuals int Dist[].der
    maxresidu=0; indresidu=0; index=0; sumresidu=0;
    for(j=0;j<LearnSamples && index<=konec;j++) {
        i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
        if(i==trida || whole>0){
            index++;
            if(index>zacatek && index<=konec && Dist[j].V>0) { //jen v
danem intervalu a nepasivovane body, ktere se ale pocitaji do "pocet"
                ri=Dist[j].D; // distance in a given Metrics
                lnIapr=C[trida]+q[trida]*log(ri); //approx in the ln
world
                residu=(double)log(index)-lnIapr; //residu in the
ln world
                Dist[j].der=residu; //lnIapr; //exp(lnIapr);
                sumresidu+=residu; //OK, truly, sum of residuals
practically equals to zero.
                if(maxresidu<fabs(residu)){
                    maxresidu=fabs(residu);
                    indresidu=j;
                }
            }
            pocet++;
        }
    }
    // nakonec mame indresidu pro pasivaci nejhorsiho bodu
    Dist[indresidu].V=0; //a pasivaci hned provedeme
    Dist[indresidu].der=0;
} //end of part RESIDUALS /**/
}

void BetaPriRegre2(TDist Dist[], double q[], double C[], double LearnArr[], long
LearnSamples, long Dimension,
    long zacatek, long konec, long Count[], long trida, long whole, long
residuals, long WEI){
    //LEPSI metoda s opakovanou lin. regresí.
    //13.1.2010 doladěna podle Fabiána
    //supposed distances sorted in ascending order
    //if whole>0 then use indexing over the whole learning set, otherwise count
samples of class "trida" only.
    long it, itmax;
    double Slnri2, Swei, Slnri, Slni, Slnilnri, weight, ri, residu, lnIapr,
sumresidu2,
        logri, logindex, W, epsvar,
inde, p2qq, sumA, sumB, sumresidu, meanRes2, meanRes2old,
        minresidu, maxresidu; //, var, varold;;
    long i, j, index, pocet, Orient, NOpq;
    char tab;

    //The Dist[j].der is used as residuals!!!, see comments.
    itmax=1000;
    epsvar=0.0001; //error criterion
    tab=9;
    NOpq=0;

    Orient=1;
    for(j=0;j<LearnSamples;j++) Dist[j].der=0; //here as initial weights
    for(it=0;it<=itmax;it++){
        //1.lin regression weighted
        Slnri2=0; Slnilnri=0; Slnri =0; Slni =0; Slnilnri=0; Swei=0;
        pocet=0; index=0;
        for(j=0;j<LearnSamples && index<=konec;j++) {
            i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
            if(i==trida || whole>0){ // if whole>0 then take all points
without respect to class.
                index++; //points of the class trida are indexed in all
cases one after another 1, 2, ...

```

```

        if(index>zacatek && index<=konec && Dist[j].V>0) { //jen v
danem intervalu
        pocet++; //count the number of truly used points for
regression (1, 2, ...)
        ri=Dist[j].D; /* distance in a given Metrics*/
        //index=(NOPq==0?ii-Orient*p2qq:ii); //Style I. -
correction of the right hand side Y.
        inde=(NOPq==0?index-Orient*p2qq:index); //Style I. -
correction of the right hand side Y.

        if(WEI==1) { //..*index/log(index) =>
W=inde==1?1:inde/log(inde); // (index+1)/log(index+1);
            logri=W*log(ri);
            logindex=(double)inde; //W*log(index);
        }
        else {
            logri=log(ri);
            logindex=log(inde);
        }
        weight=1/(Na(Dist[j].der+1,3)); //modify by weight
=1/(eps+1)^3

        //matrix
        Swei =Swei+weight;
        Slnri =Slnri+logri*weight;
        Slnri2=Slnri2+logri*logri*weight;
        //right hand side
        Slni =Slni+logindex*weight;
        Slnilnri=Slnilnri+logindex*logri*weight;
    }
} //thus we have data for weighted regression equation; index is the
last point used
q[trida]=(Swei*Slnilnri-Slnri*Slni)/(Swei*Slnri2-Slnri*Slnri);
//distribution mapping exponent
C[trida]=(Slni*Slnri2-Slnri*Slnilnri)/(Swei*Slnri2-
Slnri*Slnri); //regression constant (it estimates distribution density in the query
point)

//heuristics for strange values of Q s opravou C:
heurQC(&q[trida],&C[trida], Dimension, Dist, zacatek, konec);

//2.4.RESIDUALS: compute residuals
index=0; sumresidu2=0; pocet=0;
minresidu= 1.0e300;
maxresidu=-1.0e300;
for(j=0;j<LearnSamples && index<=konec;j++) {
    i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
    if(i==trida || whole>0){
        index++;
        if(index>zacatek && index<=konec ) { //jen v danem
intervalu
            pocet++;
            ri=Dist[j].D; // distance in a given Metrics
            lnIapr=C[trida]+q[trida]*log(ri); //approx in the ln
world
            residu=(double)log(index)-lnIapr; //Orient*(lnIapr-
(double)log(index)); //residu in the ln world
            Dist[j].der=residu; //here new residuals
            sumresidu2+=Na2(residu);
            if(it==0 && minresidu>residu) minresidu=residu;
            if(it==0 && maxresidu<residu) maxresidu=residu;
        }
    }
}
}
//end of part RESIDUALS
//orientation:
if(it==0 && maxresidu<fabs(minresidu)) Orient=-1; //once only !!

```



```

//residuals positive only !!!!!:
for(j=0;j<LearnSamples && index<=konec;j++) {
    i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
    if(i==trida || whole>0){
        index++;
        if(index>zacatek && index<=konec ) { //jen v danem
intervalu
                residu=Dist[j].der;
                if(Orient>0) residu=residu-minresidu; //Orient==1
                else residu=maxresidu-residu;
                sumA+=residu/(residu+1);
                sumB+=1/(residu+1);
                sumresidu2+=Na2(residu);
                sumresidu+=residu;
                Dist[j].der=residu;
            }
        }
    }
    meanRes2=sumresidu2/pocet; ///LearnSamples;
    p2qq=sumA/sumB; //p/qq ratio of parameters of the beta-prime
distribution [#####]

//a ven! vysledky jsou v q[trida] a C[trida]:
if(itmax==0) goto jdiven;
if(it>9){ //at least ten iterations.
    if(it>=itmax || fabs(meanRes2old-meanRes2)<epsvar) goto
jdiven; //stop criterion var-varold původně[0]
}
meanRes2old=meanRes2;

} //END of cycle variable it.
jdiven:;
} //end of BetaPriRegre2

void Regre(double LearnArr[], TDist Dist[], long Dimension, long LearnSamples, long
Count[], double q[],
        double C[], /*long whole,*/ long SEL, long WEI, long trida,
double AdHoc) {
    long i, it, zacatek, konec, whole;
    for(i=0;i<LearnSamples;i++) Dist[i].V=1; //all points are valid
    zacatek=0;
    whole=0;
    if(SEL<0) konec=-SEL;
    if(SEL==0) whole=1; //indexing over the learning set
    if(SEL==0) konec=Count[trida]; // if SEL==0 use the whole learning set.\n");
    if(SEL==1) konec=Count[trida]; // not ready yet// if SEL==1 use the whole
learning set till the crossing point.\n");
    if(SEL==2) konec=Count[trida]/2; //DEFAULT if SEL==2 use one half of samples
of the learn.set.\n");
    if(SEL==3) konec=(long)sqrt((double)Count[trida]); //if SEL==3 use first
sqrt(No. of samples of the learn.set).\n");
    if(SEL==4) {konec=Count[trida]/2; zacatek=konec/3;}
    if(SEL==5) {konec=(long)sqrt((double)Count[trida]); zacatek=konec/3;}
    if(SEL>=6) konec=SEL;

    //if(konec>LearnSamples){konec=LearnSamples; zacatek=0;} //case of forced
number of points for regression

    if(WEI==4) { //cycle of robust linear regression by deletion largest residuals
        for(it=0;it<(konec-zacatek)/2;it++){ //iterace odstranovani odchylenych
bodů na polovic počtu, který je =(konec-zacatek).
            LinRegre2 (Dist, q, C, LearnArr, LearnSamples,
Dimension, zacatek, konec, Count, trida, whole, 1, WEI); //residuals into
Dist[].der and passivate the worst
        } //konec iterace odstranovani odchylenych bodů

```

```

    }
    if(WEI==0 || WEI==1 || WEI==4)
        LinRegre2 (Dist, q, C, LearnArr, LearnSamples, Dimension,
zacatek, konec, Count, trida, whole, 0, WEI);// no residuals
    if(WEI==2) HypRegre2 (Dist, q, C, LearnArr, LearnSamples, Dimension,
zacatek, konec, Count, trida, whole, 0, WEI);// no residuals
    if(WEI==3) BetaPriRegre2 (Dist, q, C, LearnArr, LearnSamples, Dimension,
zacatek, konec, Count, trida, whole, 0, WEI);
    if(WEI==5) Takens (Dist, q, C, LearnArr, LearnSamples, Dimension,
zacatek, konec, Count, trida, whole, 0, WEI, AdHoc); //No residuals
    //heuristics for strange values of Q s opravou C počítáno prostě z prvního a
posledního bodu:
    heurQC(&q,&C, Dimension, Dist, zacatek, konec);
}

void QCrege(double LearnArr[], double OutArr[], TDist Dist[], long Dimension, long
classes, long LearnSamples, long kk,
    long Count[], double S[], double DME, long SEL, long WEI, double
AdHoc, double weit[]){
    //lin regrese pres jednotlivé tridy a bere se "aditivní" KONSTANTA, ne
směrnice
    //kk=test sample index
    long trida, i;
    double sum;
    double* q = new double [classes];

    sum=0;
    for(trida=0;trida<classes;trida++){//lin regrese pres jednotlivé tridy
        Regre(LearnArr, Dist, Dimension, LearnSamples, Count, q, S, SEL, WEI,
trida, AdHoc);
        S[trida]=exp(S[trida]);// ((double)Count[trida]; .. ORIG s delenim
pocty ve tride
    }
    sum=0;
    for(i=0;i<classes;i++) {
        OutArr[kk*classes+i]=(double)S[i]/Count[i];//spravne!! vcetne
apriornich pravdepodobnosti!! NUTNE!!
        sum+=OutArr[kk*classes+i];
    }
    for(i=0;i<classes;i++) OutArr[kk*classes+i] = OutArr[kk*classes+i]/sum;
    delete q;
}

//Crossing ggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggggg
long CrossMinCount(long classes, long Count[]) {
    long i,minCount;
    minCount=1000000000;
    for(i=0;i<classes;i++)
        if(minCount>Count[i]) minCount=Count[i];
    return minCount;
};

void CrossFill(long minCount, long classes, long Counts[], double CrossDMF[], long
trida, double S[]){
    //the last set of cells (minCount+1) in CrossDMF serves as class points
counter
    long i,j;
    i=(long) (0.01+CrossDMF[minCount*classes+trida]);
    CrossDMF[minCount*classes+trida]++;
    j=i*minCount/Counts[trida]; //recompute index to the same length of class DMF
arrays
    CrossDMF[j*classes+trida]=S[trida];
};

long CrossPosition(long Cross, long minCount, long classes, long Counts[], double
CrossDMF[], double S[]) {
    //now we have arrays of DMFs; a task is to identify the crossing point

```

```

long i,j,k,claSi,closestCl,CrossPos;
double minSi, difference;
FILE *hh;

printf("\nThe DMF:\n");
for(i=0;i<minCount+5;i++){
    for(j=0;j<classes;j++){
        printf(" %12lg",CrossDMF[i*classes+j]);
        printf("\n");
    }
}
//getchar();
/**/
for(i=0;i<minCount-1;i++){
    //find the smallest Si
    minSi=1.0e33;
    for(j=0;j<classes;j++){
        if(minSi>CrossDMF[i*classes+j]){
            minSi=CrossDMF[i*classes+j];
            claSi=j;
        }
    }
    //Find the closest class
    closestCl=-1;
    difference=1.0e33;
    for(j=0;j<classes;j++){
        if(j!=claSi){
            if(CrossDMF[i*classes+j]-minSi < difference){
                closestCl=j;
                difference=CrossDMF[i*classes+j]-minSi;
            }
        }
    }
    //Test if they cross; there already holds the first condition:
    // minSi< CrossDMF[i*classes+closestCl]
    CrossPos=0;
    //the second condition:
    if(CrossDMF[(i+1)*classes+claSi]>CrossDMF[(i+1)*classes+closestCl]){
        CrossPos=i*Cross/100;
        for(j=0;j<classes;j++){
            S[j]=CrossDMF[CrossPos*classes+j];
        }
        //Record the crossing
        if ((hh = fopen("cross.txt", "at")) == NULL){
            fprintf(stderr, "Cannot open file " "cross.txt".\n");
            exit(1);
        }
        fprintf(hh, "\nCrossing at Cross= %i from %i\n",i,minCount);
        fclose(hh);
        printf("\nCrossing found at Cross= %i from %i\n",i,minCount);
        //getchar();
        break;
    }
}
//
// getchar();
return CrossPos;
}
//Crossing end end end end end end end end end end end end

```

```

double CDglobal(double LearnArr[], double OutArr[], TDist Dist[], long Dimension,
long classes, long LearnSamples, long kkk,
    long Count[], double S[], double DME, long SEL, long WEI, double
AdHoc, double MeansDev[], long TAREsp,
    double Metrics, double CovMat[], double weit[]){
    //returns the correlation dimension computed with averaging method
    //kkk=here the number of points we take the mean for estimation of the CD
from individual DMES
    long kk, i, j, trida;

```

```

double dist2, Q, QQ;
double* q = new double [classes];

//zde pouzito AdHoc pro rizeni poctu kkk, ze ktereho se bere prumer:
if(AdHoc>kkk) kkk=(long)(AdHoc+0.5);
if(kkk>LearnSamples-1) kkk=LearnSamples-1;//ochrana pred nesmyslnym poctem

QQ=0;
for(kk=0;kk<kkk;kk++){//a global cycle over kkk cases of DME computation
using random learning samples as query points
//Our "test" sample:
for(j=0;j<Dimension;j++)
MeansDev[2*Dimension+j]=(LearnArr[kk*(Dimension+TAResp)+j]-
MeansDev[0*Dimension+j])/MeansDev[1*Dimension+j]; //recompute to zero mean and unit
sigma

//Construct Dist[.] wrt. to "test" sample and sort it
for(i=0;i<LearnSamples;i++) {//vynulujeme pole Dist
Dist[i].D=1.0e33; //inicializace vzdalenosti
Dist[i].I=i; //hodnoty indexu
};
for(i=0;i<LearnSamples;i++){//probirame radky site jeden po druhem
trida=(long)LearnArr[i*(Dimension+1)+Dimension];
dist2= distance(&LearnArr[i*(Dimension+1)],
&MeansDev[2*Dimension], CovMat, Dimension, Metrics, trida);
Dist[i].D=dist2; //Write to array Dist true distances in a given
Metrics
point
} // end of loop of learning samples for a given sample as a query
qsort(Dist, LearnSamples, sizeof(TDist), TDistCompare);
// Thus there are neighbors in array Dist of type TDist arranged
according to the distances in ascending order for given class.

//compute DME our standard way
for(trida=0;trida<classes;trida++){//lin regrese pres jednotlive tridy
WEI, trida, AdHoc);
} //?????????nejak ZDE vychazi moc velike hodnoty q !!!!!!!!
Q=0;
for(trida=0;trida<classes;trida++) Q+=q[trida];
Q=Q/classes;//the mean value of the DME
if(SEL==0) Q=2*Q;//estimation from the whole learning set
QQ+=Q;
}
QQ/=kkk;//the mean value of the DMEs
//END of CORE part ONE computing CD as mean distribution mapping exponent "Q"

delete q;
return QQ;
}

void SFSloc(double LearnArr[], double OutArr[], TDist Dist[], long Dimension, long
classes, long LearnSamples, long kk,
long Count[], double S[], double DME, long SEL, long WEI, double
DMEvalue, double AdHoc, double weit[],
long minCount, long Cross, double CrossDMF[]){
//kk=test sample index ... with weights
long trida, i;
double sum, Q, add;
double* q = new double [classes];
double* nearest = new double [classes];

if(DMEvalue>0.5) Q=DMEvalue;//use just this value as the DME
else{//compute DME our standard way
sum=0;
for(trida=0;trida<classes;trida++){//lin regrese pres jednotlive tridy

```

```

        Regre(LearnArr, Dist, Dimension, LearnSamples, Count, q, S, SEL,
WEI, trida, AdHoc);
    }
    Q=0;
    for(trida=0;trida<classes;trida++) Q+=q[trida];
    Q=Q/classes;//the mean value of the DME
    if(SEL==0) Q=2*Q;//estimation from the whole learning set
}
if(Q>=Dimension-1) Q=Dimension*3/5;//correction from 16.8.2015
//CORE part TWO - compute probability estimations for all classes for k=th
sample:
for(trida=0;trida<classes;trida++) {S[trida]=0; nearest[trida]=0;}
for(i=0;i<LearnSamples;i++){
    trida=(long) (0.001+LearnArr[Dist[i].I*(Dimension+1)+Dimension]);
    add=weit[i]/Na(Dist[i].D, Q); //!! true distances in a given Metrics
and true Q
    if(add>nearest[trida]) nearest[trida]=add;
    S[trida]+=add;
//void CrossFill(long minCount, long classes, long Counts[], double
CrossDMF[], long trida, double S[]){
    if (Cross>0) CrossFill(minCount,classes,Count,CrossDMF,trida,S);
}

//long CrossPosition(long Cross, long minCount, long classes, long Counts[], double
CrossDMF[], double S[]) {
    if(Cross>0 && CrossPosition(Cross,minCount,classes,Count,CrossDMF,S)>0){
        sum=0;
        for(i=0;i<classes;i++) {
            OutArr[kk*classes+i]=S[i];
            sum+=OutArr[kk*classes+i];
        }
    }
    else {
        sum=0;
        for(trida=0;trida<classes;trida++) {
            OutArr[kk*classes+trida]=(S[trida]-
nearest[trida])/(Count[trida]-1);
            // including apriori probabilities - necessary!!
            sum+=OutArr[kk*classes+trida];
        }
    }
    for(trida=0;trida<classes;trida++)
    OutArr[kk*classes+trida] = OutArr[kk*classes+trida]/sum;
    delete q;
    delete nearest;
}

void IINC(double LearnArr[], double OutArr[], TDist Dist[], long Dimension, long
classes, long kk,
        long Count[], double S[], long LearnSamples, double weit[],
        long minCount){
//kk=test sample index ... with weights
long i, k;
double sum, trida;

for(i=0;i<classes;i++) S[i]=0;
for(i=0;i<LearnSamples;i++) {//summing up reciprocals according to classes
    k=Dist[i].I;//index of the i-th nearest neighbor
    trida=LearnArr[k*(Dimension+1)+Dimension];//its class
    S[(long) (0.001+trida)]+=weit[k]/((double) (i+1));
//=1/((double) (i+1));
}
sum=0;
for(i=0;i<classes;i++) {
    OutArr[kk*classes+i]=(double)S[i]/Count[i];
    sum+=OutArr[kk*classes+i];
}
}

```

```

        for(i=0;i<classes;i++) OutArr[kk*classes+i]=OutArr[kk*classes+i]/sum;
    }

void Loc_k_NN(double LearnArr[], double OutArr[], TDist Dist[], long Dimension,
long classes,
        long kk, long Count[], double S[], long neighborhood, double weit[],
        long minCount, long Cross, double CrossDMF[], long LearnSamples){
//kk=test sample index, neighborhood=neighborhood size (neighborhood-NN)
long i, k, LAsamples, cl, CP, neighborh;
double sum, trida;
neighborh=neighborhood;
//Crossing
if(Cross>0){
    for(i=0;i<classes;i++) S[i]=0;
    for(i=0;i<LearnSamples;i++) {
        k=Dist[i].I;//index of the i-th nearest neighbor
        trida=LearnArr[k*(Dimension+1)+Dimension];//its class
        S[(long) (0.001+trida)]++;
        //void CrossFill(minCount,classes,Counts[], CrossDMF[], trida,
S[]){
            if (Cross>0) CrossFill(minCount, classes, Count, CrossDMF,
trida, S);
        }
        //long CrossPosition(long Cross, long minCount, long classes, long
Counts[], double CrossDMF[], double S[]) {
            CP=CrossPosition(Cross,minCount,classes,Count,CrossDMF,S);
            if(Cross>0 && CP>0 && CP<neighborhood) neighborh=CP;
        } //CrossingEnd

    for(i=0;i<classes;i++) S[i]=0;
    if(neighborh>1) { // neighborhood-NN method, neighborhood>1 weighted in
NNbase
        for(i=0;i<neighborh;i++) {
            k=Dist[i].I;//index of the i-th nearest neighbor
            trida=LearnArr[k*(Dimension+1)+Dimension];//its class
            S[(long) (0.001+trida)]++;
        }
    }
    else {// (My) 1-NN method weighted in NNbase
        LAsamples=0;
        cl=0;
        for(i=0;i<classes;i++) LAsamples+=Count[i];
        for(i=0;i<LAsamples;i++) {
            k=Dist[i].I;//index of the i-th nearest neighbor
            trida=LearnArr[k*(Dimension+1)+Dimension];//its class
            if(S[(long) (0.001+trida)]<1.0e-10){
                S[(long) (0.001+trida)]+=(Dist[i].D>1.0e-
7?1/Dist[i].D:1.0e+7);
                cl++;
            }
            if(cl==classes) break;
        }
    }

    sum=0;
    for(i=0;i<classes;i++) {
        OutArr[kk*classes+i]=(neighborhood==1?(double)S[i]:(double)S[i]/Count[i]);
        sum+=OutArr[kk*classes+i];
    }
    for(i=0;i<classes;i++) OutArr[kk*classes+i]=OutArr[kk*classes+i]/sum;
}

```

```

void CIntegral(double LearnArr[], double TestArr[], double OutArr[], TDist Dist[],
long Dimension,
    long classes, long LearnSamples, long kk,
    long Count[], double S[], double DME, long SEL, long WEI, double
AdHoc, double Metrics, double weit[]){
//kk=test sample, i.e. query point index
long i, j, k, K, L, pairs, konec, trida, CIcon, sa, CIcount, sb, s;
double dist, pom, prob, maxDist, CSvM;

if(SEL>=6) konec=SEL;
else SEL=5;
K=konec;//zacatek se nepouziva z principu
pairs=K*(K+1)/2;//the size of arrays for correlation integrals
//generate arrays for correlation integrals:
double* CI = new double [classes*pairs];
double* CI1= new double [classes*pairs];
double* nprob= new double [classes*pairs];
long* indexPrvniho= new long [classes];
//fill in arrays for correlation integrals using nearest neighbors of the
query point
for(i=0;i<pairs;i++) nprob[i]=(double)i;
for(trida=0;trida<classes;trida++) indexPrvniho[trida]=-1;
for(trida=0;trida<classes;trida++){ //Dist.D,I,
    CIcon=0;
    sa=0;
    for(j=0;j<LearnSamples && sa<K;j++){//loop over all neighbors until
        i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
        if(i==trida){//nalezen soused te tridy, ma index j.
            if(indexPrvniho[trida]==-1) indexPrvniho[trida]=j;//we do
not use the nearest neighbor now
            else {
                sb=0;
                for(k=j+1;k<LearnSamples && sb+sa<K;k++){//inner
loop over neighbors

                    i=(long) (0.001+LearnArr[Dist[k].I*(Dimension+1)+Dimension]);
                    if(i==trida){//nalezen soused te tridy, ma
index k.

                        //compute binate distance
                        dist= distance(&LearnArr[Dist[j].I*(Dimension+1)],
&LearnArr[Dist[k].I*(Dimension+1)], LearnArr, Dimension, Metrics, trida);

                        //merge as new into list of distances,
i.e build the correlation integral

                        CI[CIcon*classes+trida]=dist;
                        for(s=CIcon;s>0;s--) {
                            if(CI[s*classes+trida]<CI[(s-
1)*classes+trida]){//prehod

                                pom=CI[s*classes+trida];

                                CI[s*classes+trida]=CI[(s-1)*classes+trida];

                                CI[(s-
1)*classes+trida]=pom;

                                    }

                                CIcon++;
                                sb++;

                            }

                        }

                    }

                }

            }

        }

    }

}

//now we are ready with CI without the first point (query or the
nearest)
for(j=0;j<CIcon;j++) CI1[j*classes+trida]=CI[j*classes+trida];//copy
into CI1

```

```

        //query point has index kk in TestArr[] and its distances to all
preceding points we add to CI1 correl.integral
        CI1count=CIcount;
        maxDist=0;
        sb=0;
        for(j=0;j<LearnSamples && sb<K;j++){//inner loop over neighbors
            i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
            if(i==trida){//nalezen soused te tridy, ma index k.
                //dist=Dist[j].D;
                //if(maxDist<dist) maxDist=dist;

                //dist=0;//compute binate distance(s to all preceding
points add them to CI1 correl.integral)
                //for(s=0;s<Dimension;s++)
                //    dist+=Na2(LearnArr[Dist[j].I*(Dimension+1)+s]
                //        - TestArr[kk*(Dimension+1)+s]);
                //dist=sqrt(dist);//truly binate distance*/

                //compute binate distance
                dist= distance(&LearnArr[Dist[j].I*(Dimension+1)],
&TestArr[kk*(Dimension+1)], LearnArr, Dimension, Metrics, trida);

                //merge as new into list of distances, i.e build the
correlation integral
                CI1[CI1count*classes+trida]=dist;
                for(s=CI1count;s>0;s--) {
                    if(CI1[s*classes+trida]<CI1[(s-
1)*classes+trida]){//prehod
                        pom=CI1[s*classes+trida];
                        CI1[s*classes+trida]=CI1[(s-
1)*classes+trida];
                        CI1[(s-1)*classes+trida]=pom;
                    }
                    //else break;
                }
                CI1count++;
                sb++;
            }
        }

        //the nearest (the first) point has index L in Dist array and its
distances to all preceding points we add to CI correl.integral.
        L=indexPrvniho[trida];
        sb=0;
        for(j=0;j<LearnSamples && sb<K;j++){//inner loop over neighbors
            i=(long) (0.001+LearnArr[Dist[j].I*(Dimension+1)+Dimension]);
            if(i==trida && indexPrvniho[trida]!=j){//nalezen soused te
tridy, ma index k.

                //dist=0;//compute binate distance ...Metrics
                //for(s=0;s<Dimension;s++)
                //    dist+=Na2(LearnArr[Dist[j].I*(Dimension+1)+s]
                //        - LearnArr[Dist[L].I*(Dimension+1)+s]);
                //dist=sqrt(dist);//truly binate distance

                //compute binate distance
                dist= distance(&LearnArr[Dist[j].I*(Dimension+1)],
&LearnArr[Dist[L].I*(Dimension+1)], LearnArr, Dimension, Metrics, trida);

                //merge as new into list of distances, i.e build the
correlation integral
                CI[CIcount*classes+trida]=dist;
                for(s=CIcount;s>0;s--) {
                    if(CI[s*classes+trida]<CI[(s-
1)*classes+trida]){//prehod
                        pom=CI[s*classes+trida];
                        CI[s*classes+trida]=CI[(s-1)*classes+trida];
                        CI[(s-1)*classes+trida]=pom;
                    }
                }
            }
        }

```



```

        }
    }
    CIcont++;
    sb++;
}

//now there is a corr integral with the nearest point in CI
//      and corr.integral with the query point in CI1
//Here compute difference of corr integrals CI and CI1
//We use Cramer von Mises estimate. There are the same numbers of
points of CI and CI1
//and thus we can sum up differences of corresponding cells of CI and
CI1
//and divide the total sum by the number of cells to get, in fact, the
probability
//on the vertical axis which originally represents only counting.
//here CIcont==CI1count and should correspond to the number of points
K.

CSvM=0;
for(i=0;i<CIcont;i++) {
    //double interpolFast(double x, double xrow[], double yrow[],
long rows, long from, long interpType){
    pom=interpolFast(CI[i], CI1, nprob, CIcont, 1, 2);//nacitam
plochu mezi distribucnoimi funkce
    CSvM+=Na2((double)i-pom);
    pom=interpolFast(CI1[i], CI, nprob, CIcont, 1, 2);//nacitam
plochu mezi distribucnimi funkce
    CSvM+=Na2((double)i-pom);
}
CSvM/=(2*2*CIcont*CIcont);
prob=CSvonMis50(CSvM, CIcont);
OutArr[kk*classes+trida]=prob;//CSvM;//toto je snaha i bez prevodu na
pravdepodobnost to dostat nejak ven; jinak spravne://prob;
} //end of loop over all classes for sample No. kk in the array TestArr[kk].
//normalize to 100 %
pom=0;
for(i=0;i<classes;i++) pom+=OutArr[kk*classes+i];
for(i=0;i<classes;i++) {OutArr[kk*classes+i]=OutArr[kk*classes+i]/pom;
printf("kk=%i trida=%i %lg\n", kk, i, OutArr[kk*classes+i]);}
delete indexPrvniho;
delete CI1;
delete CI;
delete nprob;
}

//K is the neighborhood now!!!

long NNbase(double LearnArr[], double TestArr[], double OutArr[], long Dimension,
long classes,long LearnSamples,long TestSamples, long TAREsp,

double Metrics, long Repeated, double MeansDev[], long neighborhood,

double DME, long SEL, long WEI, double AdHoc, long Count[], double
weit[],

long Cross, char Message[]) {
    //result=NNbase(NN, TST, Probab, Vstupu, labels, RadkuSite, RadkuTST,
1, Metrics, 0, Means,neighborhood, DME, SEL, REG, AdHoc) ;

    //BASIC PARAMS:
    //LearnArr[LearnSamples][Dimension+1] //the learning array //C++ uklada po
radcich!!!
    //
    //      will be overridden by normalized variables!
    //TestArr[TestSamples][Dimension+TAREsp] .. the array of testing samples
    //OutArr[TestSamples] .. vector of responses to samples of the TesArr in the
same order

```

```

//ValueOfClass1 .. if desired response is equal to or larger than
ValueOfClass1 then the sample is signal
//                                     else the sample is background
//Dimension .. the number of independent variables
//classes .. true number of classes
//LearnSamples .. the number of learning samples (RadkuSite)
//TestSamples .. the number of testing samples
//TAResp .. either 0 if the TestArr does not include the desired responses
//                                     or 1 if the structure if the TestArr is the same as of
the LearnArr

//CONTROL PARAMS:
//Metrics .. metrics used for distances computation, Metrics >= 1.
//neighborhood .. the number of nearest neighbors used for class estimation.
// if neighborhood<=0 then neighborhood=sqrt(LearnSamples)
//Repeated .. if Repeated!=1 run normalization, else suppose normalized
LearnArr and valid MeansDev array
//WEI .. control of regression formula (REG)
//Cross .. if > 0 test the crossing point and reduce neighborhood accordingly.
//WORKING ARRAYS:
//MeansDev[3][Dimension] .. array of normalization constants
MeansDev[0][]=means,
//
MeansDev[1][]=std.deviation,
//
MeansDev[2][]=standardized test sample

long i, j, kk, trida, DME1, minCount;
double dist2, CDimense;
minCount= CrossMinCount(classes, Count);
double* CrossDMF = new double [(minCount+1)*classes];
double* S = new double [classes]; //the working array of partial sums for
individual classes
TDist* Dist = new TDist [LearnSamples]; //array of distances and indexes to
array of the learning set
i=1;
j=1;
if((long)(Metrics-0.001)==-1 || (long)(Metrics-0.001)==-2) i=Dimension;
if((long)(Metrics-0.001)==-2) j=classes;
double* CovMat = new double [j*i*i]; //the covariance matrix array for
Mahalanobis
double* ZZ = new double [j*i*i]; //the ZZ decomposition of covariance
matrix for Mahalanobis
double* DD = new double [j*i*i]; //the matrix of squares of eigenvalues of
covariance matrix for Mahalanobis
FILE *hh;
/*check print
long ii, jj, ss;
ss=0;
for(ii=0+ss;ii<10+ss;ii++){
for(jj=0;jj<=Dimension; jj++) printf("%10lg
",LearnArr[ii*(Dimension+1)+jj]);
printf("\n");
}
printf("Hit ENTER to continue\n");
getchar();

long ii, jj, ss;
ss=0;
for(ii=0+ss;ii<10+ss;ii++){
for(jj=0;jj<=Dimension; jj++) printf("%10lg
",TestArr[ii*(Dimension+1)+jj]);
printf("\n");
}
printf("Hit ENTER to continue\n");
getchar();
/**/

```

```

DME1=(long) (DME<0?DME-0.001:DME+0.001);//20091122

//compute normalization constants //initialize
if(Repeated!=1) //including Class Dependent Mahalanobis
    normalize(LearnArr, Dimension, LearnSamples, Count, classes, MeansDev,
Metrics);
if((long) (Metrics-0.001)==-1){// for Mahalanobis independent of class
//void CovarMatrix(double LearnArr[], long Dimension, long
LearnSamples, long Class, double CovMat[]) {
    CovarMatrix(LearnArr, Dimension, LearnSamples, -1, CovMat, Count);
//void ainvl(double z[],double d[],double a[],long n,double tol);
    ainvl(ZZ,DD,CovMat,Dimension,0.000001);
}

if((long) (Metrics-0.001)==-2){// for Mahalanobis CDM
//void CovarMatrix(double LearnArr[], long Dimension, long
LearnSamples, long Class, double CovMat[]) {
    for(i=0;i<classes;i++){
        CovarMatrix(LearnArr, Dimension, LearnSamples, i, CovMat,
Count);//&CovMat[i*Dimension*Dimension]
        ainvl(&ZZ[i*Dimension*Dimension],DD,CovMat,Dimension,0.000001);
    }
}

if(DME1==3) { //compute the correlation dimension
//CDimense=((double)Dimension)/2;
CDimense=CDglobal(LearnArr, OutArr, Dist, Dimension, classes,
LearnSamples, 100, //we use mean from 100 samples here
Count, S, DME, SEL, WEI, AdHoc, MeansDev, TResp, Metrics, ZZ,
weit);
}

//compute distances from the testing sample
for(kk=0;kk<TestSamples;kk++){//loop over all testing samples

    if(Cross>0){
        if ((hh = fopen("cross.txt", "at")) == NULL){
            fprintf(stderr, "Cannot open file
""cross.txt".\n");
            exit(1);
        }
        fprintf(hh, " %i",kk);
        fclose (hh);
    }

    printf(" %i",kk+1);//dizpet(6);
    for(j=0;j<Dimension;j++){
MeansDev[2*Dimension+j]=(TestArr[kk*(Dimension+TResp)+j]
-MeansDev[0*Dimension+j])/MeansDev[1*Dimension+j]; //recompute
to zero mean and unit sigma
//initialization of Dist array
for(i=0;i<LearnSamples;i++) {
    Dist[i].D=1.0e33; //inicialization of the distance
    Dist[i].I=i; //original indices
};
//samples one after another
for(i=0;i<LearnSamples;i++){
//compute distances
trida=(long) (0.001+LearnArr[i*(Dimension+1)+Dimension]);
dist2= distance(&LearnArr[i*(Dimension+1)],
&MeansDev[2*Dimension], ZZ,
Dimension, Metrics, trida);
//set into the Dist array and also the class mark
trida=(long) (0.001+LearnArr[i*(Dimension+1)+Dimension]);
if (dist2<1.0e-10) { //dist2<1.0e-10;//zero distance => the same
class
for(i=0;i<classes;i++) OutArr[kk*classes+i]=0;

```

```

                                OutArr[kk*classes+trida]=1.0;
                                goto hned;
                                }
                                Dist[i].D=dist2; //Write to array Dist // true distances in a
given Metrics
                                }// end of loop of learning samples for a given testing sample

                                //weighted nearest neighbor methods: we expand distances by reciprocal
of weight:
                                if(DME==-1)
                                    for(i=0;i<LearnSamples;i++)

                                Dist[i].D=Dist[i].D/(fabs(weit[i])<0.0001?0.0001:weit[i]);

                                //QUICK SORT
                                qsort(Dist, LearnSamples, sizeof(TDist), TDistCompare);
                                // Thus there are neighbors in array Dist[][trida] arranged according
the distances in ascending order for each class.
                                //END of CORE part sorting

                                for(i=0;i<classes;i++)
                                    CrossDMF[minCount*classes+i]=0;

                                //CORE part TWO - compute probability estimations for ALL classes for
kk=th sample://TTT
                                //nearest neighbor(s) method
                                if(DME1==-1) Loc_k_NN(LearnArr, OutArr, Dist, Dimension, classes, kk,
Count, S, neighborhood, weit,
                                    minCount, Cross, CrossDMF, LearnSamples);
                                //IINC method ... with weights
                                if(DME1==0) IINC(LearnArr, OutArr, Dist, Dimension, classes, kk,
Count, S, LearnSamples, weit,
                                    minCount);

                                //if(DME1==10) TIINC(LearnArr, OutArr, Dist, Dimension, classes, kk,
Count, S, LearnSamples, weit,);

                                if(DME1==1 || DME1==11) QCrege(LearnArr, OutArr, Dist, Dimension,
classes, LearnSamples, kk, Count, S, DME, SEL, WEI, AdHoc, weit);
                                //local: ... with weights
                                if(DME1==2) SFSloc(LearnArr, OutArr, Dist, Dimension, classes,
LearnSamples, kk, Count, S, DME, SEL, WEI, 0, AdHoc, weit,
                                    minCount, Cross, CrossDMF);
                                //global: ... with weights
                                if(DME1==3) SFSloc(LearnArr, OutArr, Dist, Dimension, classes,
LearnSamples, kk, Count, S, DME, SEL, WEI, CDimense, AdHoc, weit,
                                    minCount, Cross, CrossDMF);

                                if(DME1==4) CIntegral(LearnArr, TestArr, OutArr, Dist, Dimension,
classes, LearnSamples, kk, Count, S, DME, SEL, WEI, AdHoc, Metrics, weit);

                                //CORE end; in OutArr are probabilities of sample belonging to a
class
hned;;
                                howfarWhat(TestSamples, kk, Message);
                                }//end of loop over testing samples
                                printf("\n");
                                delete DD;
                                delete ZZ;
                                delete CovMat;
                                delete Dist;
                                delete S;
                                delete CrossDMF;
//                                printf("za Loc_k_NN a delete\n");
                                return 0;
}

```

```

int main(int argc, char *argv[])
{ /*NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN*/
    long i, j, k, FileType, labels, CountErr[20], Count[100], RadkuTST,
MaxClasses;
    char tab, *ptr, c = '.', paramstr[20], rovno='', ParName[20], text[20];
    double Metrics, ParValue, DME, AdHoc, a[10010];
    long sweic, Vstupu, RadkuSite, result, SEL, REG, DME1, neighborhood, Cross;
    char SouborSite[255], TestFile[255], SWEI[255];
    FILE *hh;

    printf("\n");
    if (argc<3){
        printf("IINC ver.09/2015.\n");
        printf(" with sample weights and with crossing for k-NN methods\n");
        printf("Usage: %s LearningFile TestFile [named parameters]\n\n", argv[0]);
        printf("LearningFile: each row: inputs ClassMark [sample weight]\n");
        printf(" No other data is supposed on the line.\n");
        printf(" Max. 100 classes is supposed.\n");
        printf(" Task dimension DIM and the LearningFile size are limited by the
size of\n");
        printf(" alloctable memory only.\n");
        printf("TestFile: each row: inputs [ClassMark]\n");
        printf(" Only DIM or DIM+1 items are read from the line.\n");

        printf("\nNamed parameters: Form: ParName=Parvalue\n");
        printf("FileType: a type of NN and Test files. Default=0\n");
        printf(" =0 Reading lines as numbers including ClassMark.\n");
        printf(" =1 Reading NoOfInputs numbers and Class string.\n");
        printf(" =2 Reading pairs (index of word, word count) until (-1 -1) then
Class string.\n");
        printf(" In this case state the dimension using named parameter
DIM=...\n");

        printf("SWEI: the name of the file of sample weights, eventually including
the path.\n");
        printf(" if SWEI=1 then the file of sample weights LearningFile.wei is
searched.\n");
        printf(" Default SWEI=0, i.e. no sample weights are present.\n");
        printf(" Weights should be positive only; else absolute values are
used.\n");
        printf(" The file of weights must have the same number of rows as the
learning file\n");
        printf(" If the file of weights does not have the same number of rows
as\n");
        printf(" the Learninffile error message appears and program stops.\n");
        printf("sweic: the column number in the file of smoles weights.\n");
        printf(" default 1. If the item does not exist the weight is set to 1.\n");

        printf("DIM: the task dimension when FileType 2 is used else it is stated
automatically\n");
        printf(" according to the number of items on the first row of the
LearningFile\n");
        printf("DME: selection of the method. Default 0.\n");
        printf(" if DME=-1 nearest neighbor methods. (Eventually weighted:
distance divided by weight.)\n");
        printf(" The ize of the neighborhood is given by parameter SEL.\n");
        printf(" if DME=0 IINC (1/i) method (no influence of SEL or REG).
Eventually weighted.\n");
        printf(" if DME=1 QCrege method that uses an ""additive"" constant not
DME (not weighted).\n");
        printf(" if DME=11 QCrege method with indexing over the whole
learn.set.\n");
        printf(" if DME=2 DME-local method (SFSloc7 for SEL=0). Uses DME slope.
Eventually weighted.\n");

```

```

    printf(" if DME=3 CDglobal: uses correlation dimension globally. Use
AdHoc \n");
    printf("          as the number of DMEs used for CD estimation; default
100. Eventually weighted.\n");
    printf(" if DME=4 CIntegral method; use SEL>=6 else neighborhood is
5.\n");
    printf("          Uses local comparison of CI distribution functions (not
weighted).\n");
    printf(" else default DME=0 is used.\n");

printf("SEL: selection strategy for neighborhood. Default=2.\n");
printf(" if SEL<0 use abs(SEL) nearest points.\n");
printf(" if SEL=0 use the whole learning set and q=2*q.\n");
//printf(" if SEL=1 XXX use learning set till the crossing point.\n");
printf(" if SEL=2 use one half of samples of the learn.set.\n");
printf(" if SEL=3 use first sqrt(No. of samples of the learn.set).\n");
printf(" if SEL=4 as SEL=2 starting from 1/3 of that.\n");
printf(" if SEL=5 as SEL=3 starting from 1/3 of that.\n");
printf(" if SEL>=6 use SEL nearest points.\n");
printf(" else default SEL=2 is used.\n");
getchar();

printf("REG: for DME>0 use a regression as follows.\n");
printf(" if REG=0 standard linear regression - default.\n");
printf(" if REG=1 weighted p/ln(p) linear regression.\n");
printf(" if REG=2 hyperbolic regression. NO weighting, NO robust.\n");
printf(" if REG=3 Fabian beta-prime weighted linear LMS regression.\n");
printf(" if REG=4 robust by deletion of large residuals.\n");
printf(" if REG=5 Takens CD/DME estimator (computes C too) [with ad hoc
coeff. (0.8)].\n");

printf("Metrics; default: Metrics=2.\n");
printf(" if Metrics: >= 1; Lp metrics.\n");
printf(" if Metrics: = -1; Mahalanobis metrics derived from the whole
learning set)\n");
printf(" if Metrics: = -2; Mahalanobis class dependent metrics.\n");
printf(" if Metrics: = -3; Orloci metrics.\n");
printf(" if Metrics: = -4; Angular semimetrics.\n");
printf(" if Metrics: = -5; Clark metrics.\n");
printf(" if Metrics: = -6; Lorentz metrics.\n");
printf(" if Metrics: = -7; Canberra metrics.\n");
printf(" if Metrics: = -8; Bray-Curtis (nearly) metrics.\n");
printf(" if Metrics: = -9; Intersection (nearly) metrics.\n");
printf(" if Metrics: =-10; Cayley-Klein-Hilbert metrics.\n");
printf(" if Metrics: =-11; Weierstrass metrics.\n");
printf(" if Metrics: =-12; Hassanat metrics.\n");
printf(" else Euclidean metrics.\n");

printf("Cross; default: Cross=0.\n");
printf(" if Cross = 0; Do not test the crossing point.\n");
printf(" if Cross > 0; Use the ""Cross"" percent from the crossing
point.\n");
printf("          (Cross lies between 0 and 100 percent.\n");
printf("          (Eventually, the crossing includes weights.)\n");

printf("AdHoc: a constant for fine tuninwwwwg in some cases. Default 1.\n");

printf("\nFile of results: filename = TestFile.res:\n");
printf(" The first row: lrn file name, tst file name, No. of inputs,
ValueOfClass1\n");
printf(" Other rows each:\n");
printf(" Input values (their number according to No. of inputs),\n");
printf(" class if given else 0, class found, probabilities of classes,
sample No.\n");

printf("\nAdditional files generated:\n");
printf(" register.txt:\n");

```

```

    printf("    One row is appended for each task and consists of task
description,\n");
    printf("    number of testing samples and no. of errors.\n");
    printf("    howfar.txt:\n");
    printf("    Shows percentage of samples already processed.\n");
    printf("    (good for very large data sets.)\n");

    printf("Hit ENTER to continue\n");
    getchar();
    return 0;
}
// example:          i2lrn7.txt i2tst7.txt FileType=0 SWEI=myweights.w
DME=-1 SEL=-1
// example:          lrn_fnal   tst_fnal2.txt   26   FileType=0   DME=0
Metrics=2
// example:          i2lrn7.txt i2tst7.txt   4   FileType=1     DME=2
// example:          gf2x200.txt gf-tstk4.txt 10   FileType=0     DME=3
// example:          gf2x200.txt gf-tstk4.txt 10   FileType=1     DME=2
SEL=2 Metrics=2 REG=5 AdHoc=0.8
// example:          0x.lrn      0.tst         10   FileType=2     kNN=1
// example:          i-dat.txt   i-tst.txt   33   FileType=0     kNN=1
DME=1 SEL=1 Metrics=1
// example:          tra_0_0     tst_0_0     34   FileType=1     kNN=1
// wine      tra_0_0  tst_0_0  13   FileType=1     DME=1     SEL=2 Metrics=1
// wine-test tra_0_0.015.txt  tst_0_0  13   FileType=1     DME=1     SEL=2
Metrics=1

// iris      i2lrn1.txt i2tst1.txt  4   FileType=1     DME=1     SEL=2
Metrics=1
//          rn2eeg.txt  tst2eeg.txt  2   FileType=0     kNN=-1    DME=1     SEL=3     REG=3
Metrics=1
// ParedesSynth tra-008-00.txt  tst-5001c.txt  2   FileType=1     DME=3

printf("\n%s:\n",argv[0]);
//evaluate parameters:
strcpy(SouborSite,argv[1]); SouborSite[strlen(argv[1])]=(char)NULL;
strcpy(TestFile,argv[2]); TestFile[strlen(argv[2])]=(char)NULL;
//Vstupu=strtol(argv[3],&endptr,10);

//defaults
FileType=0;
strcpy(SWEI,"0"); SWEI[1]=(char)NULL;
sweic=1;
DME=0;
SEL=2;
Metrics=2.0;
MaxClasses=100;
REG=0;
AdHoc=1;
Vstupu=-1;
Cross=0;

tab=9;

//named parameters
for(i=3;i<argc;i++){ //for aut.dimension changed from 4 to 3
    MYstrncpy(paramstr, argv[i], strlen(argv[i]));
    ptr = strchr(paramstr, rovno);
    ParValue=0;
    if (ptr) {
        MYstrncpy(ParName,paramstr,ptr-paramstr);
        sscanf(&paramstr[ptr-paramstr+1],"%lg",&ParValue);
        if(!strcmp(ParName,"FileType"))FileType=(long)ParValue;
        if(!strcmp(ParName,"SWEI")) {
            int j;
            j=strlen(argv[i]);
            MYstrcpy(SWEI, argv[i],0,5,strlen(argv[i]));
            strcpy(text, "Weighted samples"); text[16]=(char)NULL;

```

```

    }
    if(!strcmp(ParName,"sweic")) {
        sweic=(long) ParValue;
        if(sweic<1) sweic=1;
    }
    if(!strcmp(ParName,"DME")) DME=ParValue;
    if(!strcmp(ParName,"SEL")) SEL=(long) ParValue;
    if(!strcmp(ParName,"Metrics")) Metrics=ParValue;
    if(!strcmp(ParName,"REG")) REG=(long) ParValue;
    if(!strcmp(ParName,"DIM")) Vstupu=(long) (ParValue+0.01);
    if(!strcmp(ParName,"Cross")) Cross=(long) (ParValue+0.01);
    if(!strcmp(ParName,"AdHoc")) AdHoc=ParValue;
}
}

if(strcmp(SWEI,"1")==0){ //weighted case, derive name of file of sample
weights
    ptr = strrchr(SouborSite, c);
    if (ptr-SouborSite>0){
        if (ptr) MYstrncpy(SWEI,SouborSite,ptr-SouborSite);
        else MYstrncpy(SWEI,SouborSite,strlen(SouborSite));
    }
    else {strcpy(SWEI,"s");SWEI[1]=(char) NULL;}
    strcat(SWEI,".wei");
}

DME1=(long) (DME<0?DME-0.001:DME+0.001);//20091122

i=0;
if(DME1<=-1) {
    sprintf(text,"(SEL=%i)",SEL);
    strcat(text, "-NN method");
    i=1;
};
if(DME1==0) {strcpy(text, "IINC (1/i) method"); text[17]=(char) NULL; i=1;};
//if(DME1==10) {strcpy(text, "TIINC method"); text[12]=(char) NULL; i=1;};
if(DME1==1) {strcpy(text, "QCrege"); text[7]=(char) NULL; i=1;};
if(DME1==11) {strcpy(text, "QCrege-whole"); text[13]=(char) NULL; i=1;};
if(DME1==2) {strcpy(text, "DME-local"); text[9]=(char) NULL; i=1;};
if(DME1==3) {strcpy(text, "CD-global"); text[9]=(char) NULL; i=1;};
if(DME1==4) {strcpy(text, "CIntegral"); text[9]=(char) NULL; i=1;};

if(i==0) {strcpy(text, "IINC (1/i) method"); text[17]=(char) NULL; DME1=0;}
//when none of above then default

if(DME>0) {
    if(REG==0) strcat(text, " standard");
    if(REG==1) strcat(text, " weighted p/ln(p)");
    if(REG==2) strcat(text, " hyperbolic");
    if(REG==3) strcat(text, " beta-prime wei.");
    if(REG==4) strcat(text, " robust by deletion");
    if(REG==5) strcat(text, " Takens estimator");
    strcat(text, " regresion");
}
if(Cross>0) {
    //if(Cross==0) strcat(text, " No Cross");
    if(Cross>0) strcat(text, " Crossing");
    if(Cross>99) {
        strcat(text, " reduced to 50 %");
        Cross=50;
    }
    if(REG==3) strcat(text, " beta-prime wei.");
    if(REG==4) strcat(text, " robust by deletion");
    if(REG==5) strcat(text, " Takens estimator");
    strcat(text, " regresion");
}

if(strcmp(SWEI,"0")!=0) strcat(text, " sample weighted");

```



```

//here state the number of inputs:
if(FileType==2){
    if(Vstupu==-1){
        printf("Input error: You have to state the dimension using
DIM=... named parameter when using FileType=2.\n");
        return 1;
    }
}
else {
    FILE *ff;
    if ((ff = fopen(SouborSite, "rt")) == NULL){
        fprintf(stderr, "Procedure CountFile: Cannot open file to
read %s.\n", SouborSite);
        exit(1);
    }
    Vstupu=CtiRadek(ff, a)-1;//-1 for class label, works also for string
class labels.
    fclose(ff);
}

printf("%s: NNfile:%s TestFile:%s SWEI=%s sweic=%i NoOfInputs:%i FileType:%i
",
        text, SouborSite, TestFile, SWEI, sweic, Vstupu, FileType);
printf("DME=%lg SEL=%i REG=%i Metrics=%lg\n", DME, SEL, REG, Metrics);
char* labList = new char [20*MaxClasses];
//count no of samples of all classes and fill-in the NN array,
labels=0;
CountFile(SouborSite, FileType, Vstupu, labels, labList, RadkuSite, a,
MaxClasses);
double* NN = new double [RadkuSite*(Vstupu+1)];
for(i=0;i<labels;i++) Count[i]=0; //zeros to samples counts
ReadFile(SouborSite, FileType, Vstupu, labels, labList, RadkuSite, a, NN,
Count);
// tim je nactena LearnArr do pole NN[].
printf("RadkuSite=%i Labels=%i\n",RadkuSite, labels);

//reading file of weights if any
double* weit = new double [RadkuSite];
FILE *ww;
if(strcmp(SWEI,"0")!=0) {
    if ((ww = fopen(SWEI, "rt")) == NULL){
        fprintf(stderr, "Cannot open file to read %s.\n", SWEI);
        exit(1);
    }
    for(k=0;k<RadkuSite;k++){
        j=CtiRadek(ww, a);
        if(j<sweic) weit[k]=1;
        else weit[k]=fabs(a[sweic-1]);
        //printf("%10lg\n",weit[k]);
    }
    fclose(ww);//end of reading weights
}
else for(k=0;k<RadkuSite;k++) weit[k]=1;
//read file of testing data
CountFile(TestFile, FileType, Vstupu, labels, labList, RadkuTST, a,
MaxClasses);
double* TST = new double [RadkuTST*(Vstupu+1)];
double* Probab = new double [RadkuTST*labels];
ReadFile (TestFile, FileType, Vstupu, labels, labList, RadkuTST, a, TST,
Count); // tim je nacten test. soubor
printf("RadkuTST=%i Labels=%i\n",RadkuTST, labels);

/*/otisk
long ii, jj, ss;
ss=0;
for(ii=0+ss;ii<10+ss;ii++){

```

```

        for(jj=0;jj<=Vstupu; jj++) printf("%10lg  ",NN[ii*(Vstupu+1)+jj]);
        printf("\n");
    }
    printf("Hit ENTER to continue\n");
    getchar();

    for(ii=0+ss;ii<10+ss;ii++){
        for(jj=0;jj<=Vstupu; jj++) printf("%10lg  ",TST[ii*(Vstupu+1)+jj]);
        printf("\n");
    }
    printf("Hit ENTER to continue\n");
    getchar();
    /**/

    i=1;
    if(Metrics===-2) i=labels;
    double* Means = new double [i*3*Vstupu];
        //layers according to label, i.e. Class, then:
        //row[0*Vstupu+...] means of features
        //row[1*Vstupu+...] sigmas of features
        //row[2*Vstupu+...] the query point normalized

    if(SEL<0) neighborhood=abs(SEL);
        if(SEL==0) neighborhood=RadkuSite; // q=2*q.\n !!
        //if SEL=1 XXX use learning set till the crossing point.\n");
        if(SEL==2) neighborhood=RadkuSite/2; //use one half of samples of the
learn.set.\n");
        if(SEL==3) neighborhood=(long)sqrt((double)RadkuSite); //use first sqrt(No. of
samples of the learn.set).
        if(SEL==4) neighborhood=RadkuSite/2; //as SEL=2 starting from 1/3 of
that.\n");
        if(SEL==5) neighborhood=(long)sqrt((double)RadkuSite); //as SEL=3 starting
from 1/3 of that.\n");
        if(SEL>=6) neighborhood=SEL;
        if(DME1<0) neighborhood=abs(DME1);

    if(Cross>0){
        if ((hh = fopen("cross.txt", "at")) == NULL){
            fprintf(stderr, "Cannot open file ""cross.txt"". \n");
            exit(1);
        }
        fprintf(hh,"Crossing in tst %s (learn %s), Samples
tested:\n",TestFile,SouborSite);
        fclose(hh);
    }

    result=NNbase(NN, TST, Probab, Vstupu, labels, RadkuSite, RadkuTST, 1,
        Metrics, 0, Means, neighborhood, DME, SEL, REG, AdHoc, Count, weit,
        Cross, SouborSite);

    //Evaluate and write *.res file
    EvalWri(argv[0], SouborSite, TestFile, FileType, labels, Vstupu, TST,
RadkuTST, CountErr, Probab,
        labList, text, DME, SEL, Metrics);

    //Write to registering file
    for(i=1;i<16;i++) CountErr[i]=0;
    regist(argv[0], SouborSite , TestFile, RadkuTST, labels, CountErr, Metrics,
text, DME, SEL);

    return CountErr[0];
}
/*KONEC =====*/

```

Relatively universal functions

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h> // Needed for void MyRandomize() only.

typedef struct { //structure for compare procedure for sorting
    double D;
    long I;
    long V;
    double der;
} TDist;

int TDistCompare(const void *arg1, const void *arg2 ){
    //Compare both parameters for quicksort
    //qsort(array, NumOfElements, sizeof(TDist), TDistCompare );
    TDist x,y;
    x=(TDist*)arg1;//je hodnota prvnio argumentu
    y=(TDist*)arg2;
    if(x.D <y.D) return -1;
    if(fabs(x.D-y.D)<1.0e-33) return 0;
    else return 1;
}

/* nonexisting operations */

double interpolFast(double x, double xrow[], double yrow[], long rows, long from,
long interpolType){
    //Data musi byt serazena ve sloupci pro x vzestupne a nemaji byt dve hodnoty
stejne, ale když jsou, bere se pak průměr odúpovídajících yrow
    //promenna from je obvykle 0, což znamená prohledávání od začátku; jinak se
prohledává od xrow[from].
    //Typy interpolace: 1=linear, 2=Akima (not used now), ...
    long i;
    double xd,yd,xh,yh;
    //"from" out of range
    if(from < 1 || from > rows) from = 0;

    //x out of range
    if(x <= xrow[0]) {
        from = 0;
        return yrow[0];
    }
    if(x >= xrow[rows-1]) {
        from = 0;
        return yrow[rows-1];
    }
    //Linear
    //regular
    for(i = from; i<rows;i++) {
        xd = xrow[i];
        xh = xrow[i+1];
        yd = yrow[i];
        yh = yrow[i+1];
        if(x == xd && xd == xh) {
            from = i;
            return (yd + yh) / 2;
        }
        if(x >= xd && x < xh){
            from = i;
            return yd + (x - xd) * (yh - yd) / (xh - xd);
        }
    }
    from = 0;
    return xd;
}
```

```

double Na2(double x){
    return x*x;
}

double Na(double x, double n){
    double y, z;
    //long i;
    y=fabs(x);
    if(fabs(n-1)<1.0e-33) return y;
    if(log10(y)*n>230) return 1.0e100;
    z=1.0;
    //for (i=0;i<n;i++) z=z*y;
    z=exp(n*log(y));
    return z;
}

double sign(double x) {
    return (x>=0?1:-1);
}

double frac(double x){
    double y;
    y=x-floor(x);
    return y;
}

double WeibullCDF(double x,double lambda,double k){//p1=lambda=beta, p2=k=alpha p3
undefined
    return 1-exp(-Na(x/lambda,k));
}

double ParetoCDF(double x,double xm,double k){//p1=xm, p2=k p3 undefined
    return 1-Na(xm/x,k);
}

double ParetoCDFsh2zero(double x,double xm,double k){//p1=xm, p2=k p3 undefined
    return 1-Na(xm/(1+x),k);
}

//x=ADGcrit(NoRows, pist, psoll);
double ADGcrit(long n,double x[], double ff0[]){//Anderson-Darling General
criterion
//          ^^experiment    ^^teoretické, když srovnávám s
teoretickým
//          = Astar^2 in the algorithm from Wiki !!not by Scholz-Stephens
//          dle Annis-12-Anderson-Darling.PDF

    long i;
    double A2,xn;
    //
    A2=0;//dle Anderson-Darling-two-sample.pdf
    for(i=0;i<n;i++){
        //A2+=Na2(x[i]-ff0[i])/(ff0[i]*(1-ff0[i]));
        A2+=Na2(x[i]-ff0[i])/(ff0[i]*(1-ff0[i]));
    }
    //return A2;
    /*
        A2=0;//dle Annis-12-Anderson-Darling.PDF ... NEFUNGUJE - jen pro
normální rozd.

    for(i=0;i<n;i++){
        A2+=(2*i-1)*log10(ff0[i])+(2*(n-i)+1)*log10(1-ff0[i]);
    }
    A2=-n-A2/n;
    if(A2<0)A2=0;
    /**/
    xn=(double)n;

```

```

        return A2*(1+4/xn-25/(xn*xn));
    }

double ADGprob(double tkn){//Anderson-Darling General distribution
// = procento nejistoty když zamítnu shodu rozdělení
/* -----
/* If TkN exceeds the given point c=tk-l(p) then
/* reject Ho at significance level "p".
/* Input criterion tkn, output significance level (percentage)
/*! approximated to be equal or a little bit worse (larger)
double x,y;

    //tkn=0.87;//5.9694?? 1.029 0.870 0.751 0.632 0.35 0.18921944
1.992743;
    // for 0.1% 1% 2.5% 5% 10% 0.426 0.900616977
5% double side
    x=0.3845*Na(2.0*tkn+0.35, 6.3);//4.0*0.09
    x=1.0/x;
    if(x>1.0) y=1;
    else y=x;
    return y; //ADGprob=min(1, 0.09*Na(tkn+0.35; 6.3))
/* Description
If I know AD (or A-Squared), how can I calculate the Anderson-Darling Normality
Test p-value?
Solution:
Suppose asq = AD, and n = number of observations.
(it is already) Let ast = asq*(1 + 0.75/n + 2.25/(n*n))
/**/
    double ast,p;
    ast=tkn;
    if(ast>13) ast=13;
    if (0.600 < ast /* && ast < 13 */) p = exp(1.2937 - 5.709*ast +
0.0186*ast*ast);
    if (0.340 < ast && ast < 0.600) p = exp(0.9177 - 4.279*ast - 1.38*ast*ast);
    if (0.200 < ast && ast < 0.340) p = 1 - exp(-8.318 + 42.796*ast -
59.938*ast*ast);
    if (ast < 0.200) p = 1 - exp(-13.436 + 101.14*ast - 223.73*ast*ast);
    if(p>1.0) p=1.0;
    if(p<0.0) p=0.0;
    return p;
    //Reference: R.B. D'Agostino and M.A. Stephens, Eds. (1986). Goodness-of-Fit
Techniques, Marcel Dekker.
/**/
/* Description
If I know AD (or A-Squared), how can I calculate the Anderson-Darling Normality
Test p-value?
Solution:
Suppose asq = AD, and n = number of observations.
Let ast = asq*(1 + 0.75/n + 2.25/(n*n)).
-> If 0.600 < ast < 13, then p = exp(1.2937 - 5.709*ast + 0.0186*ast*ast).
-> If 0.340 < ast < 0.600, then p = exp(0.9177 - 4.279*ast - 1.38*ast*ast).
-> If 0.200 < ast < 0.340, then p = 1 - exp(-8.318 + 42.796*ast - 59.938*ast*ast).
-> If ast < 0.200, then p = 1 - exp(-13.436 + 101.14*ast - 223.73*ast*ast).
Reference: R.B. D'Agostino and M.A. Stephens, Eds. (1986). Goodness-of-Fit
Techniques, Marcel Dekker.
/**/
}

double CSvonMis50(double CSvM, long count) {
    //Cramer von Mises test of distributions
    //area = test parameter, area between distrib.functions
    //count = the second parameter of test, the number of empirical points
    //values are valid for exactly 50 empirical points counts; practical
differences for the other "counts" are small.
    //double x[18], p[18];

```

```

double x[18]={0, 0.02512, 0.03068, 0.03690, 0.04636, 0.05462, 0.06258,
0.07062, 0.11924, 0.20958, 0.24132, 0.28396,
0.346822, 0.45986, 0.57754, 0.73728, 1.14507, 2};
double p[18]={0, 0.01, 0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75, 0.8,
0.85,
0.9, 0.95, 0.975, 0.99, 0.999, 1 };
/**/
x[0]=0;          p[0]=0;
x[1]=0.02512;   p[1]=0.01;
x[2]=0.03068;   p[2]=0.025;
x[3]=0.03690;   p[3]=0.05;
x[4]=0.04636;   p[4]=0.1;
x[5]=0.05462;   p[5]=0.15;
x[6]=0.06258;   p[6]=0.2;
x[7]=0.07062;   p[7]=0.25;
x[8]=0.11924;   p[8]=0.5;
x[9]=0.20958;   p[9]=0.75;
x[10]=0.24132;  p[10]=0.8;
x[11]=0.28396;  p[11]=0.85;
x[12]=0.346822;p[12]=0.9;
x[13]=0.45986;  p[13]=0.95;
x[14]=0.57754;  p[14]=0.975;
x[15]=0.73728;  p[15]=0.99;
x[16]=1.14507;  p[16]=0.999;
x[17]=2;        p[17]=1;
/**/
//double interpolFast(double x, double xrow[], double yrow[], long rows, long
from, long interpolType)
return interpolFast(CSVm, x, p, 18, 0, 2);//[Akima]
}

double Gam(long a){ //see http://www.xycoon.com/beta_mode.htm
long i;
double x;
x=1;
for(i=1;i<=a;i++) x=x*((double)rand()/RAND_MAX);
if(x<1.0e-30) x=(double)rand()/RAND_MAX;
return log(x);
}

double BetaDist(long a, long b){ //see http://www.xycoon.com/beta_mode.htm
double GamA;
GamA=Gam(a);
return GamA/(GamA+Gam(b));
}

double NormDist01(){ //see http://www.xycoon.com/nor_random.htm
double x;
long i;
x=0;
for(i=0;i<12;i++) x+=(double)rand()/RAND_MAX;
return x-6;
}

long XXX200 (double a[], double b[], long N, double EPSILO, double *DET){
/* funkce vypocita reseni soustavy linearnich rovnic a
vraci 1 pro regularni vysledek, 0 jinak.
PIVOTY VYBIRA POSTUPNE NA HLAVNI UHLOPRICCE
A = POLE MATICE ULOZENE PO RADCICH; ZMENI SE NA JEDNOTKOVOU MATICI
B = VEKTOR PRAVYCH STRAN, KTERY SE ZMENI NA VEKTOR RESENI
N = STUPEN MATICE
EPSILO = ZADANA MINIMALNI HODNOTA PIVOTU
*DET = (POINTER NA) DETERMINAT (při volání psát &DET) */
long i,j,k;

```

```

double x;
if(N<=0) return(1); /* nesmysna dimense */
if(N<2){ /* matice je typu 1x1 */
    if(fabs(a[0])>=EPSILO) {a[0]=1./a[0]; return(1);}
    else {return(0);}
}
*DET=1.0; /* normalni matice */
for(i=0;i<N;i++){ /* hlavni krokovani po uhlopricke */
    if(fabs(a[N*i+i])<EPSILO)
        return(0); /* OSETRENI PIVOTU */
    *DET=*DET*a[N*i+i];
    x=1/a[N*i+i]; /* deleni i-teho radku pivotem */
    for(k=i;k<N;k++) a[N*i+k]=a[N*i+k]*x; /*for(k=0;...misto n^2 jenn(n-
1)/2 nasobeni*/
    b[i]=b[i]*x;
    for(j=0;j<N;j++){ /* krokovani po radcich, vynechame i-ty */
        if(j!=i){
            x=a[N*j+i]; /* prvek pro vynos. i-teho radku a jeho
odect. od radku j */
            for(k=i;k<N;k++) a[N*j+k]=a[N*j+k]-
x*a[N*i+k]; /*for(k=0;...misto n^3 jen n^2(n-1)/2 nasobeni*/
            b[j]=b[j]-x*b[i];
        }
    }
}
return(1);
}

```

```

////////////////////////////////////
////////////////////////////////////
// function [Z,D] = ainvl(A,tol); norm(inv(A)-Z*Z')-->0
void ainvl(double z[],double d[],double a[],long n,double tol){
//
// left-looking ainv decomposition
//
// purpose:
// Computes the factorization
//  $ZD^{-1}Z^T \approx \text{inv}(A)$  ??????????????
// of the symmetric and positive definite matrix A.
// simple dense code,
// where D is diagonal matrix of squares of eigenvalues
//
// input:
// A: matrix to be factored;
// tol: drop tolerance for elements in U and V factors;
//
// output:
// Z,D: AINV factors.
// returns 1 if OK else 0 if pivot is lesser than 1.0e=30.
//
// history:
// Matlab code by Mirek Tuma, 2003.
// FORTRAN code by Marcel Jirina, 2014. Needs subroutines myrow, mycol,
scas.
// c++ code by Marcel Jirina, 2014. No own externals.
//
// basic initializations
//
long i,j,k,l;
double pi,pj,pp,ajnk,zknj,xinvpp;
double* dd = new double [n];
//
// unit matrices
//
for(i=0;i<n;i++){
    for(j=0;j<n;j++) z[i*n+j]=0;
    d[i,i]=z[i*n+i]=1;
}

```

```

}
//
// the loop
//
for(i=0;i<n;i++){
    //
    // pivoting
    //
    for(j=0;j<i;j++){
        //
        // find the multipliers
        //
        pi=pj=pp=0;
        for(k=0;k<n;k++){
            pi+=(ajnk=a[j*n+k])*z[k*n+i];
            pj+=a[i*n+k]*(zknj=z[k*n+j]);
            pp+=ajnk*zknj;
        }
        //
        // update the remaining columns
        //
        if(fabs(pp)<1.0e-30) goto nic;
        xinvpp=1/pp;
        //change the i-th column of z:
        //Z(:,i) = Z(:,i) - Z(:,j)*inv(pp)*pj;
        for(k=0;k<n;k++){ //do k=1,n
            z[k*n+i]=z[k*n+i]-z[k*n+j]*xinvpp*pj;
        }
        //
        // dropping
        //
        for(k=0;k<n;k++){
            if(fabs(z[k*n+i]) < tol) z[k*n+i]=0.0;
        }
    }
    pp=0;
    for(k=0;k<n;k++) pp+=a[i*n+k]*z[k*n+i];
    d[i*n+i]=pp; //D(i,i) = pp;
    dd[i]=sqrt(1/pp);
    //
}
//
//Z=Z*sqrt(inv(D)): D is diagonal
//
for(k=0;k<n;k++){
    for(l=0;l<n;l++){
        z[k*n+l]=z[k*n+l]*dd[l];
    }
}
nic://case of singular matrix A --> z and d are then unit matrices
if(fabs(pp)<1.0e-30){
    for(i=0;i<n;i++){
        for(j=0;j<n;j++) z[i*n+j]=0;
        d[i,i]=z[i*n+i]=1;
    }
}
//return 1;
// norm(inv(A)-Z*Z')
} // of ainvl ////////////////////////////////////////

```

```

double mahala(double u[], double v[], double z[], long n){
//    v*inv(a)*u;
double vysledek;
double* dd = new double [n];
double* ls = new double [n];
long i,j;

```



```

    for(i=0;i<n;i++) dd[i]=u[i]-v[i];
    for(i=0;i<n;i++) {
        ls[i]=0;
        for(j=0;j<n;j++) ls[i]+=z[i*n+j]*dd[j];
    }
    //scalar product
    vysledek=0;
    for(i=0;i<n;i++) vysledek+=ls[i]*ls[i];
    delete ls;
    delete dd;
    return sqrt(vysledek);
} // end of mahala //////////////////////////////////////

```

```

char * MYstrncpy(char * s1, char * s2, size_t nn){
    char * k; long m, n;
    m=(long)strlen(s2);
    n=(long)nn;
    n=m>n?n:m;
    k=strncpy(s1, s2, n);
    s1[n]=(char)NULL;
    return k;
};

```

```

long MyStrCmp(char a[], char b[]){ //Porovnaní retezcu
    long i, x, y;
    x=strlen(a);
    y=strlen(b);
    if(x!=y) return 0;
    for(i=0;i<x && i<y;i++){
        if(a[i]!=b[i]) return 0;
    }
    return 1;
}

```

```

void MYstrcpy(char a[], char b[], long shiftA, long shiftB, long n){ //kopirovani
retezce b do a
    long i;
    for(i=0;i<n;i++) {
        if(b[i+shiftB]==char(NULL)) break;
        a[i+shiftA]=b[i+shiftB];
    }
    a[i+shiftA]=char(NULL);
    return;
}

```

```

void MyRandomize(){ // needs #include <time.h>
    long t, i;//time_t t;

    t = time(NULL);// 1170170074
    for(i=0;i<(t-1170170074);i++) rand();
}/**/

```

```

void dizpet(long i){/* shifts cursor on the screen i characters back */
    long j;
    char k=8;
    for(j=0;j<i;j++) fprintf(stdout,"%c",k);
}

```

```

void normalize(double LearnArr[], long Dimension, long LearnSamples, long Count[],
    long classes, double MeansDev[], double Metrics){
    long i, j;

```

```

double trida,x;
if((log)(Metrics-0.001)!=-2){//not CDM
for(j=0;j<=Dimension;j++) {MeansDev[0*Dimension+j]=0;
MeansDev[1*Dimension+j]=0;}
//compute means
for(i=0;i<classes;i++) Count[i]=0;
for(i=0;i<LearnSamples;i++){
//      printf("%6i",i+1);dizpet(6);
for(j=0;j<Dimension;j++)
MeansDev[0*Dimension+j]+=LearnArr[i*(Dimension+1)+j];
trida=0.001+LearnArr[i*(Dimension+1)+Dimension];
Count[(long)trida]++;
}
for(j=0;j<Dimension;j++)
MeansDev[0*Dimension+j]=MeansDev[0*Dimension+j]/LearnSamples;

//      for(j=0;j<Dimension;j++) printf("%lg\n", MeansDev[0*Dimension+j]);
//      getchar();
//Subtract means and compute std.deviation
for(i=0;i<LearnSamples;i++) for(j=0;j<Dimension;j++)
MeansDev[1*Dimension+j]
+=Na2(LearnArr[i*(Dimension+1)+j]-MeansDev[0*Dimension+j]);
for(j=0;j<Dimension;j++) {
x=sqrt(MeansDev[1*Dimension+j]/(LearnSamples-1));
if(x<1.0e-30) MeansDev[1*Dimension+j]=1;// the case of constants in the
learning set.
else MeansDev[1*Dimension+j]=x;
}

//      for(j=0;j<Dimension;j++) printf("%lg\n", MeansDev[0*Dimension+j]);
//      getchar();

if((long)(Metrics-0.001)==-1){//Mahalanobis independent of class
for(j=0;j<Dimension;j++) MeansDev[1*Dimension+j]=1;//std.dev not
modified
}

//Standardize the LearnArr; the array LearnArr will be overridden by
normalized variables:
for(i=0;i<LearnSamples;i++){
for(j=0;j<Dimension;j++)
LearnArr[i*(Dimension+1)+j]=(LearnArr[i*(Dimension+1)+j]
-MeansDev[0*Dimension+j])/MeansDev[1*Dimension+j];
//      printf("%lg %lg
%lg\n",LearnArr[i*(Dimension+1)+0],LearnArr[i*(Dimension+1)+1],LearnArr[i*(Dimensio
n+1)+2]);
}
//      getchar();
}
else{// CDM <=> Metrics===-2
for(j=0;j<=Dimension;j++) {MeansDev[0*Dimension+j]=0;
MeansDev[1*Dimension+j]=0;}
//compute means
for(i=0;i<classes;i++) Count[i]=0;
for(i=0;i<LearnSamples;i++){
//      printf("%6i",i+1);dizpet(6);
for(j=0;j<Dimension;j++)
MeansDev[0*Dimension+j]+=LearnArr[i*(Dimension+1)+j];
trida=0.001+LearnArr[i*(Dimension+1)+Dimension];
Count[(long)trida]++;
}
for(j=0;j<Dimension;j++)
MeansDev[0*Dimension+j]=MeansDev[0*Dimension+j]/LearnSamples;

//      for(j=0;j<Dimension;j++) printf("%lg\n", MeansDev[0*Dimension+j]);
//      getchar();

```

```

//Subtract means and compute std.deviation
for(i=0;i<LearnSamples;i++) for(j=0;j<Dimension;j++)
MeansDev[1*Dimension+j]
    +=Na2 (LearnArr[i*(Dimension+1)+j]-MeansDev[0*Dimension+j]);
for(j=0;j<Dimension;j++) {
    x=sqrt (MeansDev[1*Dimension+j]/(LearnSamples-1));
    if(x<1.0e-30) MeansDev[1*Dimension+j]=1;// the case of constants in the
learning set.
    else MeansDev[1*Dimension+j]=x;
}

//          for(j=0;j<Dimension;j++) printf("%lg\n", MeansDev[0*Dimension+j]);
//          getchar();

//if((long) (Metrics-0.001)==-2){//Mahalanobis dependent of class
for(j=0;j<Dimension;j++) MeansDev[1*Dimension+j]=1;//std.dev not
modified
//}

//Standardize the LearnArr; the array LearnArr will be overridden by
normalized variables:
for(i=0;i<LearnSamples;i++){
for(j=0;j<Dimension;j++)
LearnArr[i*(Dimension+1)+j]=(LearnArr[i*(Dimension+1)+j]
-MeansDev[0*Dimension+j])/MeansDev[1*Dimension+j];
//          printf("%lg %lg
%lg\n",LearnArr[i*(Dimension+1)+0],LearnArr[i*(Dimension+1)+1],LearnArr[i*(Dimensio
n+1)+2]);

}
//          getchar();
}
}

```

```

long EvalROC (long Samples, double Responses[], double TrueVal[], double threshold,
long OrderNo[], double SigEffi[], double BackErr[], double ClasError[], double
RejFact[], double Enrichm[], double QualFact[], double MinClasErr[]) {
/*

```

Returns Number-of-errors for min.clas.error when OK else 0 - it means less than 2 samples or samples of one class are missing.

Samples number of samples to process, i.e. the length of arrays Responses, TrueVal, and all others.
Responses one-dimensional double array of responses of the classifier; rows correspond to individual samples.
TrueVal one-dimensional double array of true class values for individual samples in the same order as in the Responses
threshold if for a sample the TrueVal is larger or equal to the threshold, the sample belongs to the signal class
 else it belongs to the background class.
OrderNo one-dimensional integer array of original order numbers of samples after processing
SigEffi one-dimensional double array of values of signal efficiency (acceptance)
BackErr one-dimensional double array of ... etc.
ClasError one-dimensional double array of ... [use items 1...(ROCcount-2) only, not 0...(ROCcount-1) !]
RejFact one-dimensional double array of ... [use items 1...(ROCcount-2) only, not 0...(ROCcount-1) !]
Enrichm one-dimensional double array of ... [use items 1...(ROCcount-2) only, not 0...(ROCcount-1) !]
QualFact one-dimensional double array of ... [use items 1...(ROCcount-2) only, not 0...(ROCcount-1) !]

```

    MinClasErr    one-dimensional double array with three cells only ... minimal
classification error [0], corresponding response or threshold [1], and area above
the ROC curve [2].
    /**/
    long p, k, CitacVzoru, pom, Sum1, Sumltot, Sum0, ROCcount;
    double ddd, SigEff, BckErr;

    // Count signals in a file and enumerate them:
    Sumltot=0;
    for(CitacVzoru=0; CitacVzoru<Samples; CitacVzoru++) {
        if(TrueVal[CitacVzoru]>=threshold) Sumltot++;
        OrderNo[CitacVzoru]=CitacVzoru+1;
    }
    //Check of valid data:
    if(!(Samples>1 && Sumltot>0 && Sumltot<Samples)) return 0;

    //Sort according to responses in ascending order by bubblesort:
    p=1; k=Samples;
    while(p==1){
        p=0; k-=1;
        for(CitacVzoru=0;CitacVzoru<k;CitacVzoru++){//internal cycle over
samples
            if (Responses[CitacVzoru+1] < Responses[CitacVzoru]){//exchange

                ddd=Responses[CitacVzoru];
                Responses[CitacVzoru]=Responses[CitacVzoru+1];
                Responses[CitacVzoru+1]=ddd;

                ddd=TrueVal[CitacVzoru];
                TrueVal[CitacVzoru]=TrueVal[CitacVzoru+1];
                TrueVal[CitacVzoru+1]=ddd;

                pom=OrderNo[CitacVzoru];
                OrderNo[CitacVzoru]=OrderNo[CitacVzoru+1];
                OrderNo[CitacVzoru+1]=pom;
                p=1;
            }
        }
    }

    //compute arrays SigEffi and BckErr and others:
    Sum1=0; Sum0=0; SigEff=0; BckErr=0;    MinClasErr[0]=1.0e33;
    for(CitacVzoru=0; CitacVzoru<Samples; CitacVzoru++){//loop over samples in
already sorted arrays
        //start of computation
        if(TrueVal[CitacVzoru]>=threshold) Sum1++;else Sum0++;
        SigEff=(double) (1.0-(double)Sum1/(double)Sumltot);
        BckErr=(double) (1.0-(double) (CitacVzoru-Sum1)/(Samples-Sumltot));
        BackErr[CitacVzoru]=BckErr;
        SigEffi[CitacVzoru]=SigEff;
        ClasError[CitacVzoru]=(1-SigEff+BckErr)/2;
        //ClasError[CitacVzoru]=((double) (Sum1+ (Samples-Sumltot)-
Sum0))/((double) Samples);
        //ClasErr = (NoOfSig + NoOfBckTot - NoOfBck) / (NoOfSigTot +
NoOfBckTot)
        if(MinClasErr[0]>ClasError[CitacVzoru]) {
            MinClasErr[0] = ClasError[CitacVzoru];
            MinClasErr[1] = Responses[CitacVzoru];
        }
        RejFact[CitacVzoru]=BckErr!=0?1.0/BckErr:0.0;
        Enrichm[CitacVzoru]=BckErr!=0?SigEff/BckErr:0.0;
        QualFact[CitacVzoru]=BckErr!=0?SigEff*sqrt(1/BckErr):0.0;
    }

    //recompute all arrays
    ROCcount=0;

    BackErr[ROCcount]=1;

```

```

        SigEffi[ROCcount]=1;
        RejFact[ROCcount]=1;
        Enrichm[ROCcount]=1;
        QualFact[ROCcount]=1;
        ClasError[ROCcount]=1;
        ROCcount++;
/**/
        for(CitacVzoru=1; CitacVzoru<Samples-1; CitacVzoru++){//loop over samples in
already sorted arrays
                if(BackErr[CitacVzoru]<BackErr[CitacVzoru-1] &&
SigEffi[CitacVzoru]>SigEffi[CitacVzoru+1]){//write next values
                        BackErr[ROCcount]=BackErr[CitacVzoru];
                        SigEffi[ROCcount]=SigEffi[CitacVzoru];
                        RejFact[ROCcount]=RejFact[CitacVzoru];
                        Enrichm[ROCcount]=Enrichm[CitacVzoru];
                        QualFact[ROCcount]=QualFact[CitacVzoru];
                        ClasError[ROCcount]=ClasError[CitacVzoru];
                        ROCcount++;
                }
        }/**/

        BackErr[ROCcount]=0;
        SigEffi[ROCcount]=0;
        RejFact[ROCcount]=0;
        Enrichm[ROCcount]=0;
        QualFact[ROCcount]=0;
        ClasError[ROCcount]=1;
        ROCcount++;
/**/
        //area below the ROC curve
        MinClasErr[2]=0;
        for(k=1; k<ROCcount; k++) MinClasErr[2]+=0.5*(BackErr[k-1]-BackErr[k])*(2-
SigEffi[k-1]-SigEffi[k]);

        return (long) (Samples*MinClasErr[0]+0.0001);
}

/* cte radek dim reals do pole a[] a vraci 0 kdyz ok, -1 pri chybe*/
long CtiRadekAlabel(FILE *gg, double *a, long dim, char *label) {
        long i;
        for(i=0;i<dim;i++){
                a[i]=0;
                if(!fscanf(gg,"%lg", &a[i])) return -1;
        }
        MYstrcpy(label, NULL, 0, 0, 0);
        if(!fscanf(gg,"%s", label)) return -1;
        return 0;
}

/* cte radek dim reals do pole a[] a vraci 0 kdyz ok, -1 pri chybe
Reading pairs (index of word, word count) until (-1 -1) then Class string.*/
long CtiRadekAlabel2(FILE *gg, double *a, long dim, char *label) {
        long i, index, indexold; double x;
        indexold=-1;//////////
        for(i=0;i<dim;i++) a[i]=0;
        for(;;){
                if(!fscanf(gg,"%i%lg", &index, &x)) return -1;
                if(index<0) break;
                if(index==indexold) return 0;
                indexold=index;
                if(index<dim) a[index]=x;
        }
        MYstrcpy(label, NULL, 0, 0, 0);
}

```

```

        if(!fscanf(gg,"%s", label)) return -1;
        MYstrncpy(label, label, 2); //MYstrncpy(char * s1, char * s2, size_t n);
        return 0;
    }

long CtiRadek(FILE *gg, double *a) { /* cte radek reals do pole a[] a vraci pocet
prectenych cisel, 0 kdyz zadne, -1 pri chybe */
    long CitUdaju, bylaMez, ii, strcCit, xx;
    char st[50000], strr[50], ch[1];
    if(!fscanf(gg,"%[^\n]\n", st)) return -1; /*přečten celý řádek jako
řetězec*/
    strcat(st, " "); //přidáme koncové mezery
    CitUdaju=0;
    strcCit=0;
    bylaMez=0;
    strcpy(ch, "");
    strcpy(strr, ""); //vyprazdñime pro nový údaj
    xx=(long)strlen(st);
    for (ii=0;ii<xx;ii++) { //cykl po jednotlivých znacích
        if(st[ii]==32 || st[ii]==9 || st[ii]==59) {
            if(bylaMez || strcCit==0) ; //když po mezeze přijde mezera
nebo tabelator, tak nic
        } else { //nebyla mezera a přišla mezera, tedy konec skupiny v
strr a to je nový údaj.
            strr[strcCit]=ch[0];

sscanf(strr,"%lg",&a[CitUdaju]); //sprintf(strr,"%lg",a[CitUdaju]);
            CitUdaju++;
            bylaMez=1;
            strcpy(strr, ""); //vyprazdñime pro nový údaj
            strcCit=0;
        }
    }
    else { //proste přišel nový znak, přidáme ho k předeslým
        if(st[ii]==44) st[ii]=46; // desetinnou čárku nahradíme
tečkou)*+,-./01789::;568<;
        MYstrncpy(&strr[strcCit],&st[ii],1);
        strcCit++;
        bylaMez=0;
        //strcpy(ch,&st[ii],1);
        //strcat(strr,ch);
    } // na konci je v CitUdaju p[ocet a v a[] jednotlivé údaje
    }
    return CitUdaju;
}

//search for label index in labList
long LabelIndex(char labList[], char label[], long labels){
    long i; char lb[20]; //LabLen
    for(i=0;i<labels;i++){
        MYstrcpy(lb, labList, 0, 20*i, 20-1);
        if(MyStrCmp(label, lb)) break;
    }
    return i;
}

//search for label in labList according to its index
void LabelByIndex(char labList[], char *label, long labels, long index){
    MYstrcpy(label, labList, 0, 20*index, 20-1);
}

void CountFile(char SouborSite[], long FileType, long Vstupu, long &labels, char
labList[], long &RadkuSite, double a[], long MaxClasses) {
    long i, j, lab, LabLen;
    char label[20], lb[20]; //LabLen

```

```

FILE *ff;

LabLen=20;
lab=labels;
if ((ff = fopen(SouborSite, "rt")) == NULL){
    fprintf(stderr, "Procedure CountFile: Cannot open file to read
%s.\n", SouborSite);
    exit(1);
}

if(FileType==1 && lab==0) {
    for(i=0;i<MaxClasses*LabLen;i++) labList[i]=(char)NULL;
}
//labels=0;
RadkuSite=0;
for(i=0;;i++){ //for(i=0;i<100000;i++){
    printf("%6i",i+1);dizpet(6);

    if(FileType==0) {
        j=CtiRadek(ff, a);// here I need number of labels "labels"
        sprintf(label,"%lg", a[Vstupu]);
    }

    if(FileType==1){
        if(0>CtiRadekAlabel(ff, a, Vstupu, label)) return;
    }

    if(FileType==2){
        if(0>CtiRadekAlabel2(ff, a, Vstupu, label)) break;//
    }

    if(MyStrCmp(label, "")) goto EndLrnFile1;
    RadkuSite++;

    if(lab==0){
        if(labels==0){strcpy(labList, label); labels++;}
        else {
            for(j=0;j<labels;j++){
                MYstrcpy(lb, labList, 0, 20*j, 19);
                if(MyStrCmp(label, lb)) {
                    break;
                }
            }
            if(j==labels) {//new label
                MYstrcpy(labList, label, 20*labels, 0,
                strlen(label));
                if(labels>=MaxClasses) {
                    printf("Procedure CountFile:
Exceeded max number of labels possible (%i).\n", MaxClasses);
                    return;
                }
                labels++;
            }
        }
    }

    if(feof(ff))
        break;
}
EndLrnFile1;;
fclose(ff);
//RadkuSite=i+1;
}

```

```

void ReadFile(char SouborSite[], long FileType, long Vstupu, long &labels, char
labList[], long &RadkuSite, double a[], double NN[], long Count[]){
//    ReadFile(SouborSite, FileType, Vstupu, labels, LabList, RadkuSite, a, NN);
    long i, j, tst, LabLen;
    char label[20]; //LabLen
    FILE *ff;

    tst=Count[0]; //when reading testing file, we already have nonzero samples
count according to labels.
    LabLen=20;
    if ((ff = fopen(SouborSite, "rt")) == NULL){
        fprintf(stderr, "Cannot open file of Network.\n");
        exit(1);
    }

    for(i=0; i<RadkuSite; i++){
        printf("%6i", i+1); dizpet(6);

        if(FileType==0) {
            j=CtiRadek(ff, a); // here I need number of labels "labels"
only
            if(j<Vstupu+1)
                goto EndLrnFile2;
            sprintf(label, "%lg", a[Vstupu]);
        }

        if(FileType==1){
            if(0>CtiRadekAlabel(ff, a, Vstupu, label)) return;
        }

        if(FileType==2){
            if(0>CtiRadekAlabel2(ff, a, Vstupu, label)) return;
        }

        if(tst==0) { //learning file. We must count samples according to
labels.
            if(MyStrCmp(label, "")) goto EndLrnFile2;
            for(j=0; j<=Vstupu; j++){ //we must save all
inputs and the output!
                NN[i*(Vstupu+1)+j]=a[j];
            }
            //long LabelIndex(char labList[], char label[20], long
labels){
                j=LabelIndex(labList, label, labels);
                NN[i*(Vstupu+1)+Vstupu]=j;
                Count[j]++;
            }
            else { //testing file need not have labels
                if(MyStrCmp(label, "")) a[Vstupu]=-9; //when there are no
labels in the testfile
                else a[Vstupu]=LabelIndex(labList, label, labels);
                for(j=0; j<=Vstupu; j++) NN[i*(Vstupu+1)+j]=a[j];
            }

            iffeof(ff) break;
        }
    }
    EndLrnFile2::
        fclose(ff);
}

int FileRowsCount(FILE *ggg) {
    //if OK returns No.of rows of given file from the point where it stays
    //else returns -1.
    int i;
    char *st;

```



```

    st=new char[5000];

    for(i=1;;i++){ //for(i=1;i<100000000;i++){
        if(!fscanf(ggg,"%[^\\n]\\n", st)) return -1; //read the whole row as a
string
        if(feof(ggg)) break;
    }
    delete st;
    return i;
}

```

```

int FillArray(FILE *gg, double *Arr, int rows, int cols, int emptyRows){
    //reads rows organized file to array
    //at the beginning it jumps over emptyRows lines
    //if ok returns true No. of rows, else -1
    int i, j, n;
    char *st;
    double *pA;

    st=new char[500];
    pA = new double[cols+10];
    for(i=0;i<emptyRows;i++){
        if(!fscanf(gg,"%[^\\n]\\n", st)){//read the whole row as a string
            i=-1;
            goto x1;
        }
        if(feof(gg)){
            i=-1;
            goto x1;
        }
    }
    for(i=0;i<rows;i++){
        n=CtiRadek(gg, pA);
        if(n==-1){
            printf("Error filling array in FillArray procedure\\n");
            i=-1;
            goto x1;
        }
        for(j=0;j<cols;j++){
            if(j<n) Arr[i*cols+j]=pA[j];
            else Arr[i*cols+j]=0;//fill-in zeros if necessary
        }
        if(feof(gg)) break;
    }
x1:
    delete pA;
    delete st;
    return i;
}

```

```

double Sigmoida(double x, double MinZ, double MaxZ){
    double y,x1,aa, sigm;
    aa=fabs(MinZ)+fabs(MaxZ);
    if ((aa > 1.0E+20)|| (fabs(aa-1111)<1.0e-10))
        sigm = x;
    else {
        if (fabs(aa-2222)<1.0e-10)
            sigm = sqrt(fabs(x))*sign(x);
        else {
            if (fabs(aa-3333)<1.0e-10) {
                if (fabs(x-0)<1.0e-10)
                    sigm = 0;
                else
                    sigm = exp(log(fabs(x))/3)*sign(x);
            }
        }
    }
}

```

```

        else {
            x1 = x*2/(MaxZ - MinZ); /*prepocet na long. <-1 +1>.*
            if (x1 > 80) x1 = 80;
            if (x1 < -80) x1 = -80;
            y = exp(-x1);
            y = (1-y)/(1+y);
            sigm = y * (MaxZ - MinZ)*0.5 + 0.5*(MinZ+MaxZ);
        }
    }
    return sigm;
}

double InvSigmo(double x, double MinZ, double MaxZ) {
    double y, aa;
    aa=fabs(MinZ)+fabs(MaxZ);
    if ((aa > 1.0E+20) || (fabs(aa-1111)<1.0e-10)) return x;
    else{
        if (fabs(aa-2222)<1.0e-10) return x*x*sign(x);
        else {
            if (fabs(aa-3333)<1.0e-10) return x*x*x;
            else {
                y = 2 * (x - 0.5*(MinZ+MaxZ)) / (MaxZ - MinZ);
                if (y > 0.999999) y = 0.999999;
                if (y < -0.999999) y = -0.999999; /*sigmoida je zde oboustranna, tj -1 az
+1*/
                y = 0.5*log( (1+y)/(1-y) ); /*interval <-1 +1>.*
                return y*0.5*(MaxZ - MinZ);
            }
        }
    }
}

void EvalWri(char program[], char SouborSite[], char TestFile[], long FileType,
long labels,
            long Vstupu, double TST[], long RadkuTST, long CountErr[],
double Probab[], char labList[],
            char text[], double DME, long SEL, double Metrics){
    FILE *gg;
    char tab, c, SouborRes[255], *res=".RES", *ptr, labela[20],
labelXresult[20];
    long i, j;
    double prob, found, soll;
    //generate response file *.res
    tab=9;
    c='.';
    ptr = strrchr(TestFile, c);
    if (ptr-TestFile>0){
        if (ptr) MYstrncpy(SouborRes,TestFile,ptr-TestFile);
        else strcpy(SouborRes,TestFile);
    }
    //else strcpy(SouborRes,"TestFile");
    else strcpy(SouborRes,TestFile); //20150804
    strcat(SouborRes,res);
    //Open response file *.res
    if ((gg = fopen(SouborRes, "wt")) == NULL){
        fprintf(stderr, "Cannot open file of Results.\n");
        exit(1);
    }
    if(Metrics>0)
        fprintf(gg, "%s : %s SEL=%i, lrn file=%s, tst file=%s,
inputs=%i, FileType=%i, Metrics=%lg\n",
            program, text, SEL, SouborSite,TestFile,Vstupu,FileType,
Metrics);
    else //for e.g. Bayes approach not using any metrics in data space
        fprintf(gg, "%s : %s SEL=%i, lrn file=%s, tst file=%s,
inputs=%i, FileType=%i\n",
            program, text, SEL, SouborSite,TestFile,Vstupu,FileType);
}

```

```

//Evaluate and write results
CountErr[0]=0;
for(i=0;i<RadkuTST;i++){
    soll=TST[i*(Vstupu+1)+Vstupu];

    prob=0.0; found=0;
    for(j=0;j<labels;j++) {
        if(Probab[i*labels+j]>prob) {

            prob=Probab[i*labels+j];//hodnota maxima
            found=(double)j; //misto
(index=trida) maxima
        }

        if(fabs(found-soll)>0.1) CountErr[0]++;

        for(j=0;j<Vstupu;j++) fprintf(gg,
"%lg%c",TST[i*(Vstupu+1)+j], tab);
        LabelByIndex(labList, labela, labels,
(long)(0.001+soll));
        LabelByIndex(labList, labelXresult, labels,
(long)(0.001+found));
        fprintf(gg, "%s%c%s%c", labela, tab, labelXresult, tab);
        for(j=0;j<labels;j++) fprintf(gg,
"%lg%c",Probab[i*labels+j], tab);
        fprintf(gg, "%i\n", i+1);
        printf(      "%s%c%s%c%i\n", labela, tab, labelXresult,
tab, i+1);
    }
    fclose(gg);
}

void regist(char program[], char SouborSite[], char TestFile[], long RadkuTST, long
labels, long CountErr[],
    double Metrics, char text[], double DME, long SEL){
//registers into register.txt file by append:
//either errors found (the first cell of CountErr is nonzero, the other zero)
//or for cases with output meaning the probability the No. of cases above the
following 16 probability levels:
//0.40 0.42 0.44 0.46 0.48 0.50 0.52 0.54 0.56 0.58 0.60 0.62 0.64 0.66 0.68
0.7
//The Nos. of cases are stored in CountErr array
FILE *hh;
char tab;
long i, sum;
if ((hh = fopen("register.txt", "at")) == NULL){
    fprintf(stderr, "Cannot open file ""register.txt"". \n");
    exit(1);
}
tab=9;
sum=0; for(i=0;i<16;i++) sum+=CountErr[i]!=0?1:0;
fprintf(hh, "%s%c%s%cSEL=%c%i%c%s%c%s%c%i%c%5.2lf%c%i",
program, tab, text, tab, tab, SEL, tab, SouborSite, tab, TestFile,
tab,
labels, tab, Metrics, tab, RadkuTST);
printf(      ("%s%c%s%cSEL=%c%i%c%s%c%s%c%i%c%5.2lf%c%i",
program, tab, text, tab, tab, SEL, tab, SouborSite, tab, TestFile,
tab,
labels, tab, Metrics, tab, RadkuTST);
if(sum<1) sum=1;//to print even if there is zero error
for (i=0;i<sum;i++){
    fprintf(hh, "%c%i", tab, CountErr[i]);
    printf (      "%c%i", tab, CountErr[i]);
}
}

```

```

        fprintf(hh, "\n");
        printf (    "\n");
    fclose(hh);
}

void howfar(long RadkuTST, long radek){
    //Checks how the program run has proceed
    //writes into hawfar.txt file by append:
    //percentage of work, program name, total testing rows, rows already
    computed,
    FILE *hh;
    char tab;
    double percent;
    if ((hh = fopen("howfar.txt", "wt")) == NULL){
        fprintf(stderr, "Cannot open file ""howfar.txt"". \n");
        exit(1);
    }
    tab=9;
    percent=100*((double)radek+1)/(double)RadkuTST;
    fprintf(hh, "%7.3lf %%, i.e. %c%i of %c%i",
            percent, tab, radek+1, tab, RadkuTST);
    fprintf(hh, "\n");
    fclose(hh);
}

void howfarWhat(long RadkuTST, long radek, char Text[]){
    //Checks how the program run has proceed
    //writes into hawfar.txt file by append:
    //percentage of work, program name, total testing rows, rows already
    computed,
    FILE *hh;
    char tab;
    double percent;
    if ((hh = fopen("howfar.txt", "wt")) == NULL){
        fprintf(stderr, "Cannot open file ""howfar.txt"". \n");
        exit(1);
    }
    tab=9;
    percent=100*((double)radek+1)/(double)RadkuTST;
    fprintf(hh, "%s:%c%7.3lf %%, i.e. %c%i of %c%i\n",
            Text,tab,percent, tab, radek+1, tab, RadkuTST);
    //    fprintf(hh, "\n");
    fclose(hh);
}

/*END =====*/

```