Institute of Computer Science
The Czech Academy of Sciences

The IINC System under the ROOT Environment

Marcel Jiřina

Technical report No. V-1253

October 2017

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 033 50, fax: +420 286 585 789,
e-mail: marcel@cs.cas.cz

Institute of Computer Science
The Czech Academy of Sciences

The IINC System under the ROOT Environment

Marcel Jiřina [1]

Technical report No. V-1253

October 2017

Abstract:

In this report we describe the interface of the IINC classification system to other programs in the c++ in the ROOT environment. The gist of this report forms a detailed description of individual c++ functions that form the IINC for the ROOT system. A basis of the three methods of the IINC system is explained. We also discuss the classification ability of the IINC system and influence of the distance function used. The use of individual methods of the system and selection of a distance function are discussed. An excerpt from literature devoted to the IINC methods is attached in the Appendix as well as results of some tests with tasks from the UCI Machine Learning Repository.

Keywords:
IINC; Data separation; Classification; Multivariate data; Distance; Metric

[1]Institute of Computer Science, The Czech Academy of Sciences, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic, E-mail: marcel@cs.cas.cz

Contents

1. Introduction

The target of this work is to show how the IINC system can be used under the ROOT environment. The whole problem is to transform ntuples of root into simple two-dimensional double data arrays and vice versa. For this purpose the IINC system contains several c++ functions that can be easily called from any c++ program. These functions are of the following three types.

- There are functions that have names of two-dimensional arrays and more or less control parameters as formal parameters.
- For the use of these functions under the ROOT environment, there are functions that provide an interface between ROOT n-tuples and the functions above. The approach used here is very simple and can be considered as an example. A ROOT user can construct his own version to conform to his particular needs.
- Other functions use file names instead of names of arrays and read corresponding text files. We suppose that functions of the first and second type mentioned can be useful in application in the field of particle physics.

Note 1. In this report we denote the classification system as IINC. It consists of three methods, one of them we denote "iinc", the other two "Local" and "Global".

2. IINC methods

The IINC methods belong among simple classifiers (data separators), eventually regressors, like the nearest neighbor rules 1-NN and k-NN or the naive Bayes classifier. IINC methods are distance-based as 1-NN and k-NN. Thus, distance plays a crucial role and is given by a distance function, mostly metric in the $\mathbb{R}^d$. Another crucial role is played by the effective dimension of the data space. This is unknown for 1-NN and k-NN methods. Depending on a particular IINC method, the effective dimension may be a correlation dimension, a local scaling factor, or an approximation of scaling behavior of data. Then, there are three IINC methods

- "Global" method that uses the correlation dimension ν supposing that data are monofractal, i.e. a fractal nature, scaling, is approximately the same over the entire data space.

- "Local" method supposes that data are multifractal, i.e. scaling is different in a different part of the data space. Local method uses the local scaling exponent. This exponent is called a distribution mapping exponent, DME. Note that mean of local DMEs is equal to the correlation dimension.
- "iinc" method - Inverted Indexes of Neighbors Classifier estimates a probalility of a class of an event or sample (a query point) according to a proper sum of reciprocals of indexes of neighbors of this event; the nearest neighbor has index 1, the second nearest has index 2 etc. It can be shown that reciprocals of indexes model scaling features of data and, at the same time, correct errors caused by random placement of neighbors close to the query point.

The first two methods must state the correlation dimension or the distribution mapping exponent before a core procedure is used. So, there exists a learning phase in them.

Note 2. Methods of the IINC system ere reminiscent of the 1-NN and k-NN methods. But the use of the notion of a distance is all they have in common, compare [2] with any of [11], [12], [13], [10].

Each of the three methods above has its advantages and disadvantages.

- An advantage of the global method is that the correlation dimension one has to estimate ones only for a data set, i.e. for a particular problem consisting of several or many events to be classified. Another advantage is that correlation dimension can be estimated sufficiently well. A disadvantage lies in the fact that correlation dimension may differ from the local scaling exponent, the distribution mapping exponent, and this may be a source of errors. This method uses distances and its complexity is proportional to the size of the learning set (not considering the learning phase, i.e. the estimate of the correlation dimension).
- The local method should be (theoretically) most exact among all three methods. The DME gives the value of scaling on the spot of the query point and leads to the best estimate of probability of a class to which the query point belongs. In practice, it is not true. The estimate of the distribution mapping exponent is made by linear regression or robust regression but the large spread of points nearest to the query point causes errors in the estimate of the DME and thus

classification errors. This method has complexity also proportional to the size of the learning set.

- The iinc method uses ranking of points of the learning set according to the distance from the query point. Ranking brings a similar advantage that ranking statistic has over standard statistic. Small variations are hidden in the same rank and eventual large changes influence the rank. An advantage lies in a kind of robustness of this method that also brings the lowest classification error compared to the other two methods and also compared to other classification approaches. The complexity of this method for a large data set is given by $N \log N$, where N is the size of the learning set.

The behavior of all three methods depends on the distance function, usually a metric in the $\mathbb{R}^d$. In the IINC system there are 25 different distance functions. Most of them are metrics in mathematical sense. The other are pseudometrics, i.e. they do not conform to the triangle unequality. It appears that a rather strange Hassanat metric is the best. Also an L1 (taxicab or Mannhattan) metric is very good. The Euclidean metric L2 can be considered acceptable. On the other hand, IINC with a distance function that generally (let us say on average) is not good enough may still work best for a particular task.

3. IINC system

The IINC system is written in c++, some parts use the ROOT environment. The core part forms a program iinc.cpp (iinc.h) containing the very basic (core) function iincbase().

3.1. IINC for ROOT

```
xxxxxxxxxxx
  ntiinc.h
xxxxxxxxxxx
     includes fiinc.h, iinc.h
     consists of a function
```

3.1.1. *FUNCTIONS TO CALL - WITH DATA IN NTUPLE FILES*

```
****************************************
void ntiinc(char* problemheading, char* ntlrnfile, char* ntlrn,
     char* nttstfile, char* nttst, char* ntoutfile, char* ntout){
Reads learning and testing data from root ntuple files and writes
     output data into an ntuple file.
Parameters: nt files. All other parameters are defaults:
               two classes (separator), iinc method, Hassanat metric,
               no sample weights.
```

3.2. IINC functions

```
xxxxxxxxxx
 iinc.h
xxxxxxxxxx
     consists of functions:
```

3.2.1. *FUNCTIONS TO CALL - HELP FILES*

```
*******************************************
     void iinchelp()
     void iincbasehelp()
          Both these functions call void commonhelp().
```

3.2.2. *FUNCTIONS TO CALL - WITH DATA IN TEXT FILES*

```
****************************************
void iincmain(char SouborSite[], char TestFile[], long FileType, long Vstupu,
     double Metric, long Repeat, double DME, long SEL,
     long REG, char SWEI[], long sweic,
     long Cross, char txt2[], long OUTL, double AdHoc, char argv0[], long RUM) {
The function of the IINC system that reads files and is controlled
     by other parameters that must be known at moment of run,
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
void iincmain6p(char *SouborSite, char *TestFile,
     double Metric, double DME, long SEL, long REG) {
The function of the IINC system that reads files and is controlled
```

```
     by other four parameters that must be known at moment of run.
     In fact, this function calls iincmain function with some parameters constant.
     It allows to control the Metric, the iinc method (DME), part selected for
          estimation (SEL), and type of regression (REG). There are defaults,
          namely two classes (separator) and no sample weights.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
void iincmain2p(char *SouborSite, char *TestFile) {
The function of the IINC system that needs names of files only,
     In fact, this function reads files to arrays, then calls iincbase function,
     writes *.res file and appends a row to register.res file.
     It uses defaults, namely
          two classes (separator), iinc method, Hassanat metric,
          no sample weights.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

3.2.3. *FUNCTIONS TO CALL - WITH DATA IN double ARRAYS*

```
*******************************************
long iincbase(double *LearnArr, double *TestArr, double *OutArr, long Dimension,
     long classes,long LearnSamples,long TestSamples, long TAResp,
     double Metric, long Repeated, double DME, long SEL,
     long WEI, double AdHoc, long *Count, double *weit,
     long Cross, char *Message, long OUTL) {
The very basic (core) function of the IINC system.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
long iincbase14(double *LearnArr, double *TestArr, double *OutArr,
     long Dimension, long LearnSamples,long TestSamples, long TAResp,
     double Metric, double DME, long SEL, long REG,
     long *Count, double *weit, char *Message) {
The basic function that performs all computations with the number of parameters
     reduced to 14.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
long iincbase9(double *LearnArr, double *TestArr, double *OutArr,
     long Dimension, long LearnSamples, long TestSamples, long TAResp,
```

```
     long *Count, double *weit) {
The basic function that performs all computations with the number of parameters
     reduced to 9.
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
```

3.3. Demo and first steps

The use of the IINC under the ROOT environment means using ROOT structures. We suppose input data and results in the form of ntuples.

There are two ntuples as inputs, the learning set and testing set or raw data (without known class). The learning set has form of a set of rows. Each row corresponds to an event and consists of a constant number of parameters and a class mark, usually 0 or 1. The testing set has the same structure. In case of "raw" data, the class mark is missing. The output ntuple gives probabilities of belonging to class 0 and to class 1 in the same order as there are events in the testing set.

Here is a simple example how to call function ntiinc(). All that the program does is reading ntuples from files, calling the IINC and then writing results into an ntuple. You may find a very different way to do this. You can use any function mentioned above. In case of problems, contact the author of this report.

```
xxxxxxxxxxx
  ntiinc.h
xxxxxxxxxxx

//A simple example how to use ntiinc data separator
#include "stdio.h"
#include "ntiinc.h"
#include "TNtuple.h"

int main(){//NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
char dest[255], dest1[255], dest2[255], dest3[255], dest4[255], dest5[255], dest6[255];

//preamble: It is supposed that data are in ntuple files.
```

```
//If data are in text files you have to get text files into ntuples
txtfileTOntfile (strcpy(dest,"i2lrn7.txt"), strcpy(dest1,"ntuple1.nt"),
    strcpy(dest2,"ntlrn1"), strcpy(dest3,"ntMyHeading"),strcpy(dest4,
    "f1:f2:f3:f4:class"));
txtfileTOntfile (strcpy(dest,"i2tst7.txt"), strcpy(dest1,"ntuple2.nt"),
    strcpy(dest2,"nttst1"),strcpy(dest3,"ntMyHeading"),strcpy(dest4,
    "f1:f2:f3:f4:class"));
```

3.3.1. *CALLING THE IINC SYSTEM*

```
// *****************************************
ntiinc(strcpy(dest,"MyTaskOne"), strcpy(dest1,"ntuple1.nt"),
     strcpy(dest2,"ntlrn1"), strcpy(dest3,"ntuple2.nt"),
     strcpy(dest4,"nttst1"), strcpy(dest5,"ntout.nt"), strcpy(dest6,"ntout"));
//Reads learning and testing data fom root ntuple files and writes output
//   data into a ntuple file.
//void ntiinc(char* problemheading, char* ntlrnfile, char* ntlrn,
//   char* nttstfile, char* nttst, char* ntoutfile, char* ntout){
//problemheading      your mame of the problem
//ntlrnfile, ntlrn    file and name of the ntuple containing the learning set
//nttstfile, nttst    file and name of the ntuple containing the testing set.
//   The testing set need not contain a class number (0, 1)
//ntoutfile, ntout    file and name of the ntuple containing probabilities
//   of class 0 and class 1.

//closing:
//If you wish, you can get ntfile to a text file
ntfileTOtxtfile(strcpy(dest,"ntout.nt"), strcpy(dest1,"ntout"), strcpy(dest2,"iincoutfile"));

//or you can show results on screen using e.g.
ntfileTOscreen(strcpy(dest,"ntout.nt"), strcpy(dest1,"ntout"));

printf("\nReady.\n");
return 0;
}
```

4. Discussion

It is supposed that the ML system is able to set up its intrinsic parameters according to examples presented during the learning phase, and then it is able to solve a task in an optimal way. Usually the learning phase is rather time-consuming. The IINC system has a tiny learning phase, if any. It relies on an important feature of data, their effective dimension. This notion says how "dense" are data from a topological point of view. Note that data in $\mathbb{R}^d$, having the effective dimension equal to the dimension d, are totally random. They carry no useful information and are also known as Poisson's process. Two of the three methods of the IINC system compute and use the effective dimension. In the third method ("iinc") the effective dimension is hidden in the algorithm and need not be computed explicitly.

The IINC is distance-based. Then it depends on the distance function used. There are 25 different distance functions in the system, mostly metrics. Some of them are less known and look strange. By this we mean that they do not conform to our feeling how a geometric object should look like or what features it should have. For example, we accept that a ball in L_1 metric has the form of a "diamond" and in L_∞ metric that of a cube with edges parallel with coordinate axes. These "balls" have a finite volume. But there are metrics where a "ball" has an infinite volume at least for some finite radii [4]. On the other hand, some of the strange metrics give a very low classification error in some tasks.

When using a larger set of tasks, one can rank different distance functions as shown in Table in Fig. 2 (Appendix C). But detailed inspection of this table shows, on average, that some distance functions that are not by far the best can be the best for a particular task. Thus, to find the best distance function for a particular set of similar tasks one should test all distance functions. One may doubt if the best distance function thus found is really the best. Fortunately, the difference in the classification error for the best and the second best, eventually the third best, is usually very low.

5. Conclusion

The IINC system is a program for data classification (separation) based on the notions of local and global scaling exponents known from the theory of fractals. The

purpose of this work is to show how one can use the IINC system under the ROOT environment. Only one particular approach is shown here. The use of ntuples allows straightforward calls of several functions that form the IINC system. Of course, one may find other ways of communication between ROOT and IINC.

In three appendices one can find basic facts from the IINC theory, some results of tests with various data from the UCI MLR [14], and IINC help in explaining forms of input data and control variables. Note that default values can be used for all control variables.

6. Acknowledgment

This work was supported by the Czech Ministry of Education, Youth and Sports in project No. LG15047 Cooperation on experiments at the Fermi National Laboratory, USA.

Appendices

Appendices contain an IINC help divided into several parts, and short excerpts from papers dealing with iinc and results of some tests.

Appendix A. IINC help

This help contains description of all parameters for control of the IINC system. Most of them are set up to some proven value.

A.1. HELP - EXPLANATION OF CONTROL VARIABLES

```
DIM: the task dimension (when FileType 2 is used - not here) 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. (Event. weighted: distance divided by
            weight.)
            The size of the neighborhood is given by parameter SEL.
  if DME=0  IINC (1/i) method (no influence of SEL or REG). Ev. weighted.
            Good first choice is Metric=-12 (Hassanat), -15 (H2 metric),
            -16 (H3 pseudometric), eventually -21 (Pearson) or 1 (L1).
  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.
            Good first choice is Metric=1, SEL=0 and REG=0.
  if DME=3  CDglobal: uses correlation dimension globally. Use AdHoc as the
            number of DMEs used for CD estimation; default 100. Ev.weighted
            Good first choice is Metric=1, SEL=4 and REG=0.
  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=1 use the whole learning set.
```

```
  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.

REG: for DME>0 use a regression as follows.
  if REG=0 standard linear regression - default.
  if REG=3 Fabian beta-prime weighted linear LMS regression.
  if REG=5 Takens CD/DME estimator (computes C too) [with ad hoc coeff. (0.8)].
  if REG=6 Fibonacci section method minimizing sagginess in Count=f(r^q)
coordinates.
  if REG=7 Huber's robust regression.
  if REG=8 Tucker's robust regression.

Metric; default: Metric=2.
  if Metric: >= 1; Lp metric.
  if Metric: = -1; Mahalanobis metric derived from the whole learning set)
  if Metric: = -2; Mahalanobis class dependent metric.
  if Metric: = -3; Orloci metric.
  if Metric: = -4; Angular semimetric.
  if Metric: = -5; Clark metric.
  if Metric: = -6; Lorentz metric.
  if Metric: = -7; Canberra metric.
  if Metric: = -8; Bray-Curtis (nearly) metric.
  if Metric: = -9; Intersection (nearly) metric.
  if Metric: =-10; Cayley-Klein-Hilbert metric.
  if Metric: =-11; Weierstrass metric.
  if Metric: =-12; Hassanat metric.
  if Metric: =-13; Hyperbolic metric.
  if Metric: =-14; Elliptics metric.
  if Metric: =-15; H2 metric.
  if Metric: =-16; Product semimetric (H3).
  if Metric: =-17; Polynomial distance semimetric (Ha22).
```

```
  if Metric: =-21; Pearson (nearly) metric.
  if Metric: =-22; Jacknife (nearly) metric.
  if Metric: =-23; Goodman-Kruskal (nearly) metric.
  if Metric: =-24; Kendall metric.

Repeat: if equal to 1 then do not use normalization; default 0.
Cross: if 1 then supress crossing phenomenon especially for k-NN method;
default 0.
txt2: any text.
OUTL: searching for outliers txt2: any text.
argv0: any text, but preferable the name of the program (argv[0]).
RUM: reduce the learning set to RUM randomly selected samples for each class
(no weights is supposed)
  RUM must not be greater then the number of samples of the smallest class.
AdHoc: a constant for fine tuning in some cases. Default 1.
```

A.2. HELP FOR FUNCTIONS TO CALL - WITH DATA IN double ARRAYS

For control variables see A.1.

```
LearnArr: a one-dimensional double array with rows (ie. events) one after
another where
  each row contains: variables and Label, i.e. ClassMark (i.e. 1 for signal, 2
for background sample.
  No other data is supposed on the line.
  Task dimension DIM and the LearningFile size are limited by the size of
  alloctable memory only.
TestArr: each row: inputs [Label, i.e. ClassMark]
```

A.2.1. *HELP FOR FUNCTIONS TO CALL - WITH DATA IN TEXT FILES* For control variables see A.1.

```
LearningFile: each row: inputs ClassMark
  No other data is supposed on the line.
  Max. 100 classes is supposed.
  Task dimension DIM and the LearningFile size are limited by the size of
```

```
  alloctable memory only.
TestFile: each row: inputs [ClassMark]
  Only DIM or DIM+1 items are read from the line.
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=...

Named parameters: Form: ParName=Parvalue

SWEI: the name of the file of sample weights, eventually inluding 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 samples weights.
  default 1. If the item does not exist the weight is set to 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 text files generated:
  register.txt:
    One row is appended for each task and consists of task desription,
    number of testing samples and no. of errors.
  howfar.txt:
```

```
    Shows percentage of samples already processed.
    (good for very large data sets.)
  CT.txt:
    Task parameters and a contingency table is appended to file CT.txt if it
exists.
A long run of the program can be interrupted without loss of results computed
  meanwhile with the use of text file ""ctrl"" with first character ""s"" or
""S"".
```

Appendix B. IINC methods

B.1. Multidimensional data in R^d

B.1.1. *Data space.* Let the data set U of total $N_U = N(U) < \infty$ samples be given. Each sample $x_t = \{x_{t1}, x_{t2}, \dots x_{td}\}; t = 1, 2, ...N_U, x_{tk} \in R; k = 1, 2, ..., d$ corresponds to a point in d-dimensional space $\mathbb{R}^d$, where d is the sample space dimension. For each $x_t \in U$ output $y \in \mathbb{R}$ is introduced.

In the case of a classifiers, output $y = c$, i.e an output equals to a mark, a class function $T : R^d \to 1, 2, ...\mathcal{C} : T(x_t) = c; c \in 1, 2, ..., \mathcal{C}$ With the class function the set U is decomposed into disjoint classes $U_c = \{x_t \in U \mid T(x_t) = c\}; U = \cup_{c=1}^{C} U_c, U_c \cap U_b = \emptyset; c, b \in 1, 2, ..., \mathcal{C}; c \neq b$. Let the cardinality of set U_c be $N_c; \sum_{c=1}^{C} N_c = N_U$.

Usually data are represented as a matrix with d+1 columns and N rows. Each of columns 1 till d corresponds to a feature, the column No. $d + 1$ contains an output or a class mark. Each row corresponds to one sample, point, pattern, eventually event that consist of values of d features and an output. Generally, there is no ordering of rows, i.e. of samples; they follow one after another randomly and are indexed 1, 2, .. N. Samples are often called points as each sample can be viewed as a point in a d-dimensional space.

B.1.2. *Learning and testing set* There is a different terminology used in literature. Here we speak about the learning set used for setting up the recognition system, and the testing set (checking set) "never seen before" for evaluation of true recognition capability. The learning set can be divided into the training set really used in setting up the recognizer and validating set for validation of the previous setting cycle. In the

following the validation set will have a different role.

B.1.3. *Indexing data and neighbors' distances* As we need to express which sample is closer or further from some given sample x, we can rank samples of the learning set according to distance r_i of sample x_i from sample x. Therefore, let samples of U be indexed (ranked) so that for any two samples $x_i, x_j \in U$ there is $i < j$ if $r_i < r_j; i, j = 1, 2, ...N$, and output y or class $U_c = \{x_i \in U | T(x_i) = c\}$. Of course, ranking depends on sample x and eventually of a distance function of $\mathbb{R}^n$.

From now on we use numbering of samples according to their order as neighbors of sample x; x_i being the i-th nearest neighbor of sample x.

B.2. IINC system description

A theoretical basis of the IINC classifier is described in [13]. It belongs into simple distance-based classifiers. For classification into two classes and the same size of the learning set for both classes, i.e. the same a priori probability it holds

$$p(c|x) = \lim_{i \to \infty} \frac{\sum\limits_{x_i \in U_c} 1/i}{H_N}. \tag{1}$$

where i is the index (rank) of the i-th nearest neighbor x_i of point x (without considering the neighbor's class) and $p(c|x)$ is the probability that point x belongs to class c.

In practice and for c classes one uses the following procedure: Let the samples of the learning set (i.e. all samples irrespective of the class) be sorted according to their distances from the given point x. Let indexes be assigned to these points so that 1 is assigned to the nearest neighbor, 2 to the second nearest neighbor of the given point x etc. N_c is the number of samples of class c (cardinality of U_c), $c = 1, 2, ...C$. The estimate of the probability that point x belongs to class c is

$$\hat{p}(c|x) = \frac{\frac{1}{N_c} \sum\limits_{x_i \in \cup_c} 1/i}{\sum\limits_{k=1}^{C} (\frac{1}{N_k} \sum\limits_{x_j \in U_k} 1/i)} \tag{2}$$

For total N samples and single given point x the procedure consists of three steps: Computation of distances; the computational complexity for one distance is proportional to dimensionality d, of all N distances dN. Sorting distances is

proportional to $N \log N$. Summing up of reciprocals of indexes is proportional to N. Then the total complexity is $a_1 dN + a_2 N \log N + a_3 N = N(a_1 d + a_2 \log N + a_3)$, where a_1, a_2, a_3 are implementation dependent constants. For larger learning data sets the complexity is governed by sorting. It is also seen that the computational complexity directly depends on the learning set size N and in small extend on dimensionality d.

B.3. The exponent in a classification method

Informally, consider partial influences of individual points to the probability that point x is of class c. Each point of class c in the neighborhood of point x adds a little to the probability that point x is of class c, where $c = \{0, 1\}$ is the class mark. This influence is the larger the closer to point x the point considered is, and vice versa. A question arises where do these considerations come from? The answer is simple: Let us consider a (fixed) query point x. Construct a ball $B(x, r)$ with center x a$V(x, r) = S_n r^n$. S_n is a constant dependent on space dimensionality n. Let there be i points in $B(x, r_i)$. We are interested in the density of points in the neighborhood of point x. It is $p(x, r_i) = \frac{i}{V(x,r_i)} = \frac{i}{K r_i^n}$, where K is a dimension dependent constant. Considering this density constant (in the sense that we get the same density for different values of i, i.e. for different radii r_i), radii r_i would grow proportionally to the n-th root of i, $r_i \approx \sqrt[n]{i}$. But as shown in [3], [1] the space can have (locally or globally) effective dimensionality q lesser than n. Thus, we should compute density of points using the volume of q-dimensional ball using formula, where S_q is a constant. Taking one isolated point only in distance r_i, $i = 1, 2, ...$, we get formulae for the first, second,... point. Individual points are independent; and then we can sum up probabilities above. Thus, we add the partial influences of k individual points together by summing up

$$\hat{p}(c|x) = \sum_{x_i \in U_c} \Pr(T(x) = c | T(x_i) = c) \tag{3}$$

Note that the sum goes over indices i for which the corresponding samples of the learning set are of class c, $c = 1, 2, ?C$, where C is the number of classes. It can be seen that any change of distance r_i of any point i from point x will influence the probability that point x is of class c. It can also be considered as a kernel method with kernels located at all points x_i, $i = 1, 2, ..$ in distances r_i. We can rewrite (3) in a more suitable

form for practical computation.

$$\hat{p}(c|x) = \frac{\sum\limits_{x_i \in U_c} 1/r_i^q}{\sum\limits_{i=1}^{N} 1/r_i^q} \tag{4}$$

The upper sum goes over indices i for which the corresponding samples of the learning set are of class c. At the same time all N points of the learning set are used as a normalization term. Moreover, we often do not use the nearest point ($i = 1$). It can be found that its influence is more negative than positive on the probability estimate. It holds [11]

Theorem 1. *Let the task of classification into two classes be a given. Let the size of the learning set be N and let both classes have the same number of samples. Let $q, 1 < q < n$ be the effective dimension, let i be the index of the i-th nearest neighbor of point x (without respect to class), and $r_i > 0$ its distance from point x. Then,*

$$\hat{p}(c|x) = \lim_{N\to\infty} \frac{\sum\limits_{x_i \in U_c} 1/r_i^q}{\sum\limits_{i=1}^{N} 1/r_i^q} \tag{5}$$

is probability that point x belongs to class c.

Note that the convergence of sums in the formula above is the faster the larger effective dimension q is. Usually, for multivariate real-life data the DME is also large (and the correlation dimension as well).

B.4. Three methods

Modifying considerations above, one can find that there are three methods according to what kind of data we consider and the way we deal with ratio $1/r^q$.

- If data are multifractal, then variable q differs for different places of data space. In [9], [13] it is called the distribution mapping exponent and it is a kind of effective local dimension. In this case it is necessary to compute q for each query point extra. The apparent advantage is a more exact estimate than in the "global" approach below that uses, in fact, the mean of all q's over the whole learning set.
- For monofractal data, there is $q = \nu$; q has a constant value in the data space and equals to the correlation dimension. The advantage is that we need to compute

correlation dimension ν only once. There are many methods leading to a rather exact estimate of the correlation dimension.

- IINC (Inverted Indices of Neighbors Classifer) is based on the observation explained below. The method needs no q or ν estimation, on the other hand, it uses sorting.

B.4.1. *Multifractal Data Classifier* This classifier uses the distribution mapping exponent, i.e. a local effective dimension q. Its advantage in more exact estimate of the distribution mapping exponent q may appear as an disadvantage not only for repeated computation of q for each query point separately, but also for large error in its estimate. It is often given by a relatively small amount of data especially in cases of small learning data set. Fortunately, it can be shown that the method is not too sensitive to exactness of stating the q. In any case one must check the estimated value of q for cases where q is too close to data space dimensionality n or even larger than n, or too close to zero. In these cases it is advisable to use q estimated for another point already computed and near to the query point or to compute q for several near points, exclude cases now discussed and take a mean. Many similar approaches can be designed including estimate of correlation dimension for small subset of the learning set consisting of points near to the query point.

B.4.2. *Monofractal Data Classifier* Beside the advantage that we need to compute correlation dimension ν only once, there are two problems related to it. First, the correlation dimension estimation can be time-consuming for large learning data sets because one must compute all $N(N-1)/2$ distances between points of all pairs of points of the learning set. Second, do we know if our data are monofractal? To answer these questions we show here the sensitivity of classification error to error in correlation dimension estimation, and the fact that the spread of the distribution mapping exponent is relatively small. According to [10] we can conclude that data are generally multifractals as the scaling exponent, DME varies from point to point of the set. On the other hand, these variations usually lie in a rather narrow band and thus mean DME, i.e. the correlation dimension ν, may suffice for characterization of the fractal nature of data.

B.4.3. *IINC - Inverted Indices of Neighbors Classifier* This is a modification of the first (multifractal) method above. Based on observation that the distribution mapping function grows linearly in log(*count*) - log(*distance*) coordinates with slope q and thus linearly in coordinates *count* vs. $(distance)^q$. So, the index of the neighbor i is proportional to r_i^q. Then, $1/r_i^q \propto 1/i$. From this follows that we can use the neighbor's index i instead of its distance to the q-th (or ν-th) power, r_i^q. Rewriting (4), we get

$$\hat{p}(c|x) = \frac{\sum\limits_{x_i \in U_c} 1/i}{\sum\limits_{i=1}^{N} 1/i} \tag{6}$$

This approach is very simple and very effective; no estimate of the singularity (fractal) exponent is needed.

Note 3. All three approaches, and especially this last one, are suggestive of kernel methods but with a rather strange kernel that is neither positive definite, nor finite. The use of kernels for classification and regression is beyond any discussion; support vector machines are subject of many papers and books. Here we point out several basic facts. First, there is a kernel trick that allows computation of (relatively simple) kernel function instead of search for representing features, computation a nonlinear transform from samples (pattern) space into feature (kernel) space, and then a scalar product evaluation. As a basis of the support vector machine, the learning error is minimized via quadratic programming problem. To get a unique solution, there is a condition of positively definite Gram matrix, and then kernel should be positively definite. This is a nearly standard demand. Generally, the kernel need not be necessary positively definite. Also, the kernel need not be finite. In [25] it is shown nicely that perceptron is also a kernel machine and sigmoid kernel is nowhere positively definite. In other words, we want to say that in the context of the method we present here these usually very basic and prohibitive conditions need not be studied.

In the case of IINC method, there is a close relation to the Zipfian distribution (Zipf's law). The simplest case of Zipf's law is a "$1/f$ function". Given a set of Zipfian distributed frequencies of occurrence of some objects, sorted from the most common to the least common, the second most common frequency will occur 1/2 as often as the first. The third most common frequency will occur 1/3 as often as the first, and

so on. Over fairly wide ranges, and to a fairly good approximation, many natural phenomena obey Zipf's law. Note that in the case of a "$1/f$ function", N must be finite and its denominator is equal to H_N, the so-called harmonic number, i.e. the sum of truncated harmonic series; otherwise the denominator is a sum of harmonic series, which is divergent.

Note 4. Nearly the same formula as (6) can be used for regression tasks as follows

$$\hat{y} = \frac{\sum\limits_{j=1}^{N} \frac{y_j}{rank(x_j)}}{\sum\limits_{j=1}^{N} \frac{1}{rank(x_j)}} \tag{7}$$

(In words, we compute reciprocals of the rank of the distance of point x to all points x_j; the smallest distance corresponds to rank one. Then we multiply them by corresponding outputs in the numerator. In the denominator is sum of the reciprocals only.)

In a similar way one can modify (4).

B.5. Generalization

Formulas (4), and (6) need some generalization since there are several classes C and a different number of samples N_c, $c = 1, 2, ..., C$ of each class in the learning set. General formulas are as follows:

$$\hat{p}(c|x) = \frac{1}{N_c} \frac{\sum\limits_{x_i \in U_c} 1/r_i^q}{\sum\limits_{k=1}^{C} \left(\frac{1}{N_k} \sum\limits_{x_i \in U_k} 1/r_i^q \right)} \tag{8}$$

(Eventually $q = \nu$),

$$\hat{p}(c|x) = \frac{1}{N_c} \frac{\sum\limits_{x_i \in U_c} 1/i}{\sum\limits_{k=1}^{C} \left(\frac{1}{N_k} \sum\limits_{x_i \in U_k} 1/i \right)}. \tag{9}$$

We can see the introduction of relative representation of different numbers of samples N_c of each class, i.e. introducing a priori probabilities.

B.6. Metrics in $\mathbb{R}^d$

A natural metric in $\mathbb{R}^d$ is Euclidean metric and $(\mathbb{R}^d, \rho_E)$ is a metric space. At the same time one uses another metric ρ_x. This metric we use for stating distances

between points in $\mathbb{R}^d$.

The IINC system comprises about 25 different metrics and distance functions. All distance functions that are not metrics are pseudometrics, i.e the triangle unequality can be violated in some cases.

In [5] it was shown that the iinc classifier with the Hassanat metric gives the best performance among all other distance functions used there. In concordance with this we found that other good metrics are "h2" - simplified Hassanat metric, L1 (taxicab or Manhattan) metric, Pearson, Orloci, angular metrics. The Euclidean metric L2 was found the seventh best from 24 distance functions tested.

Note 5. The metric according to Hassamat [4] is a rather strange metric defined as follows. Let $M_i = max(x_i, y_i)$ and $m_i = min(x_i, y_i)$. The metric is given by formula

$$\rho_H = \frac{1}{d}\sum_{j=1}^{d} d_i \tag{10}$$

where

$$d_i = 1 - \frac{1 + m_i}{1 + M_i}$$

for $m_i >= 0$, and

$$d_i = 1 - \frac{1 + m_i + |m_i|}{1 + M_i + |m_i|}$$

that is also

$$d_i = 1 - \frac{1}{1 + M_i + |m_i|}$$

for $m_i < 0$.

It has been proved [4] that it is really metric on $\mathbb{R}^d$ and thus $(\mathbb{R}^d, \rho_H)$ is a metric space.

In (10) according to [4] is a sum, not an average, but from context it is apparent that this is an misprint in the paper.

Appendix C. Application in classification problems

Fifteen databases have been used for the classification task into two to 10 classes. The number of attributes not including the class mark differs from 4 to 180. Data originally from the UCI Machine learning repository were gained mostly from [17] (denoted by P in column Source in the Table). These data sets are ready for run with

a classifier. Each task consists of 50 pairs of training and testing sets corresponding to 50-fold cross validation. For DNA data a single partition into training and testing sets according to specification in the UCI MLR was used. We also added the popular Iris data set. We use them without Setoza class, i.e. with two classes Versicolor and Virginica only, and the remaining data were split into 10 pairs for ten-fold cross validation.

C.1. Results of tests.

Summary of measurements is shown in the Table in Fig. 2. Rows in the table shows results for individual tasks, the last line gives the mean. Sixteen columns give classification errors for all tasks for sixteen distance functions, mostly metrics, with the IINC classifier. We selected all metrics that give error at most by 50 % worse that the best one. Thus Clark, Goodman-Kruskal, Kendall, Bray-Curtis, hyperbolic, intersection, Cayley-Klein-Hilbert, amd jacknife distance functions available in the iinc were excluded from further consideration. Columns of the table are ranked according to the mean classsification error, the best leftmost. The last column gives classification errors for the SVM (support vector machine), implementation by T. Joachims [8], [7]. Data for other classifiers, 1-NN and k-NN type with or without learning one can find e.g. in [18], [5].

Inspecting table in Fig. 2, especially the first and the last rows, one can see that

- No distance function is the best for each task.
- Considering all 25 distance functions available in the iinc the mean of smallest errors for each of 24 tasks according to Table in Fig. 1 is 12.32% and the mean of smallest errors for each of five best distance functions (see Table in Fig. 2) is 13.92%. At the same time there are 10 tasks from 24 for which one of the best five distance functions is also the best of all 25 distance functions available.
- Generally the best is the Hassanat metric. It is then set as a default metric. From five best distance functions the Hassanat metric is the best for 7 tasks. Next four distance functions (h2, L1, Ha22, Pearson) are best in other 3, 5, 5, and 4 tasks, respectively.

Dataset	Dimension (# attributes)	Number of classes	Total samples	Learning set size	Test set size	Cross validation	Source
Australian	42	2	690	551	139	50	P
Balance	4	3	625	499	126	50	P
Cancer	9	2	683	546	137	50	P
Diabetes	8	2	768	614	154	50	P
DANN	180	3	31186	2000	1186	1	P2
German	24	2	1000	800	200	50	P
Glass	9	6	215	169	46	50	P
Heart	25	2	270	216	54	50	P
Ionosphere	34	2	351	280	71	50	P
Iris (1)	4	2 (3)	100 (150)	90	10	10	UCI MLR
Led17	24	10	2000	1595	405	50	P
Letter	16	26	20000	16000	4000	1	UCI MLR
Liver	6	2	345	276	69	50	P
Monkey1	17	2	556	444	112	50	P
Phoneme	5	2	5404	4322	1082	50	P
Satimage	36	7	6435	4435	2000	1	UCI MLR
Segmen	19	7	2310	1848	462	50	P
Sonar	60	2	208	165	43	50	P
Vehicle	18	4	846	675	171	50	P
Vote	16	2	435	347	88	50	P
Vowel	10	11	528	418	110	50	P
Waveform21	21	3	5000	3998	1002	50	P
Waveform40	40	3	5000	3999	1001	50	P
Wine	13	3	178	141	37	50	P

FIGURE 1: Table of basic characteristics of tasks from UCI Machine Learning Repository modified by sources cited. Abbreviations for sources: P - [17]; P2 - [16]; UCI MLR - [14]. Note (1): Iris data are used without Setoza class, i.e. two classes Versicolor and Virginica only.

mean error	15.68%	15.85%	15.98%	16.02%	16.10%	16.13%	16.13%	16.55%	18.58%	19.22%	20.44%	21.10%	23.16%	23.31%	28.34%	31.12%	22.72%
Metric	Hassanat	h2 simpl. Hass.	L1	Ha22	Pearson	Orloci	Angular	L2	L10	CDMaha	Spearman	Weier strass	Lorentz	Mahala nobis	Canberra	elliptic	xx
Method	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	iinc	SVM
Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	BEST
australian	12.9%	12.6%	13.3%	13.0%	15.2%	15.2%	15.2%	14.8%	22.3%	18.0%	12.7%	15.9%	15.1%	31.4%	16.7%	20.5%	36.0%
balance	31.4%	34.3%	32.5%	31.1%	10.7%	23.7%	23.7%	30.5%	30.2%	30.0%	26.8%	29.6%	17.6%	30.6%	17.6%	23.6%	33.2%
cancer	3.2%	3.6%	3.3%	4.0%	6.3%	2.9%	2.9%	3.5%	3.9%	4.0%	9.9%	4.6%	6.2%	4.0%	5.6%	34.3%	16.3%
diabetes	27.1%	26.7%	26.2%	25.8%	34.5%	26.8%	26.8%	25.5%	27.1%	26.0%	35.7%	36.8%	38.8%	28.5%	39.1%	40.1%	29.6%
DNA	27.5%	27.5%	27.8%	28.2%	16.9%	16.6%	16.6%	31.0%	34.8%	17.6%	15.3%	30.9%	27.0%	46.9%	26.9%	32.3%	0.0%
german	29.5%	29.9%	30.9%	30.1%	32.6%	32.7%	32.7%	31.1%	31.2%	33.4%	31.5%	29.7%	31.3%	36.4%	29.3%	33.5%	27.3%
glass	30.4%	30.8%	33.0%	34.2%	32.5%	32.6%	32.6%	35.2%	37.9%	46.9%	31.5%	47.0%	46.2%	65.5%	49.1%	38.8%	32.6%
heart	17.9%	17.9%	18.0%	17.5%	18.4%	18.5%	18.5%	17.9%	21.9%	18.7%	18.8%	19.3%	18.8%	35.2%	20.1%	29.5%	37.2%
ionosphere	8.6%	8.1%	10.8%	11.0%	12.2%	12.4%	12.4%	14.8%	14.0%	14.8%	11.6%	32.4%	14.2%	14.7%	9.7%	35.8%	18.5%
iris	7.9%	7.9%	7.9%	5.9%	3.9%	4.9%	4.9%	4.9%	4.9%	4.0%	32.8%	4.0%	9.9%	4.0%	9.9%	37.6%	5.5%
led17	0.5%	0.5%	0.5%	0.5%	4.5%	5.1%	5.1%	0.4%	0.1%	10.4%	17.1%	0.5%	0.8%	0.6%	79.9%	5.5%	11.5%
letter	5.1%	5.5%	4.9%	5.3%	6.8%	6.2%	6.2%	5.0%	7.2%	7.2%	12.0%	12.7%	38.9%	7.3%	39.0%	15.2%	2.7%
liver	37.0%	37.9%	38.3%	40.4%	36.8%	36.5%	36.5%	39.1%	41.9%	35.7%	37.1%	50.8%	50.5%	38.7%	50.9%	46.3%	35.5%
monkey1	4.8%	4.8%	4.8%	4.8%	4.9%	5.6%	5.6%	4.8%	6.4%	9.6%	23.8%	8.0%	4.9%	20.9%	21.4%	2.5%	2.9%
phoneme	16.7%	16.8%	17.6%	17.4%	19.3%	18.6%	18.6%	18.1%	18.4%	18.7%	30.1%	32.9%	32.3%	18.2%	32.3%	28.8%	14.4%
satimage	11.3%	11.5%	11.0%	11.7%	15.7%	11.6%	11.6%	11.6%	13.5%	13.3%	19.5%	16.0%	35.3%	12.5%	35.4%	26.5%	24.3%
segmen	3.7%	4.1%	4.1%	4.5%	5.0%	5.3%	5.3%	5.0%	6.7%	5.0%	7.0%	16.0%	26.5%	6.0%	26.5%	45.3%	46.5%
sonar	20.9%	20.8%	19.9%	19.6%	23.9%	22.7%	22.7%	22.8%	29.3%	34.7%	22.7%	22.2%	20.2%	24.7%	35.9%	40.0%	19.7%
vehicle	28.8%	28.9%	29.4%	29.2%	28.9%	29.7%	29.7%	29.3%	31.0%	36.5%	29.6%	31.2%	40.4%	44.4%	40.8%	38.2%	28.2%
vote	8.9%	8.9%	8.5%	8.7%	9.3%	9.7%	9.7%	8.9%	10.5%	13.9%	9.0%	8.9%	8.8%	14.0%	8.3%	26.0%	22.6%
vowel	2.8%	2.9%	2.7%	3.3%	3.9%	3.8%	3.8%	2.7%	4.2%	13.0%	10.4%	17.2%	30.2%	4.3%	30.2%	34.8%	13.6%
waveform21	16.7%	16.8%	16.1%	16.3%	17.9%	18.3%	18.3%	16.4%	16.9%	21.0%	17.8%	16.0%	18.2%	20.5%	19.4%	37.7%	26.9%
waveform40	17.7%	17.9%	17.6%	17.3%	21.1%	21.3%	21.3%	18.1%	23.4%	22.0%	20.9%	17.6%	18.0%	21.1%	27.8%	40.1%	32.2%
wine	4.9%	4.0%	4.2%	4.9%	5.5%	6.5%	6.5%	5.7%	8.3%	6.9%	6.9%	6.2%	5.8%	29.0%	8.1%	33.9%	27.8%
worse than	Hassanat	1.11%	1.87%	2.13%	2.63%	2.80%	2.80%	5.27%	15.61%	18.42%	23.30%	25.71%	32.30%	32.74%	44.68%	49.63%	31.00%

FIGURE 2: Table of classification errors of the IINC classifier with 16 metrics for 24 tasks. The last column shows results obtained using SVM with best kernel for each task.

- The L1 (taxicab or Manhattan metric) is the second most popular metric. Fortunately, it appears to be by 1.87 % worse than Hassanat metric.
- The L2 - Euclidean metric gives mean classification error by 5.27 % worse than the Hassanat metric and by 3.46 % worse than L1 metric. So, if you prefer a "common" over "unusual" (metric), then you can use L1 metric with success.

References

[1] Camastra,P., Vinciarelli,A. (2001), Intrinsic Dimension Estimation of Data: An Approach based on Grassberger-Procaccia?s Algorithm. *Neural Processing Letters* Vol. 14, No. 1, pp. 27-34.

[2] Cover, T. M., Hart, P. E. (1967), Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, Vol. IT-13, No. 1, pp. 21-27.41

[3] Grassberger, P., Procaccia, I. (1983), Measuring the strangeness of strange attractors, *Physica*, Vol. 9D, pp. 189-208.

[4] Hassanat, A.B. (2014), Dimensionality Invariant Similarity Measure. *Journal of American Science*, Vol. 10, No. 8, pp. 221-226.

[5] Hassanat, A.B.. Abbadi, M.A., Altarawneh, G.A., Alhasanat, A.A. (2014), Solving problem of K parameter in the KNN classifier using an ensemle learning approach. *Internationa Journal of Computer Science and Information Security (IJCSIS)*, Vol. 12, No. 8, pp. 33-39.

[6] Hornik, K. (1991), Approximation capa bilities pof multilayer feedforward network. *Neural Networks*, Vol. 4, pp. 251-257.

[7] Joachims, T. (1999), Making Large-Scale SVM Learning Practical. *In: Advances in Kernel Methods - Support Vector Learning, eds. B. Scholkopf, C. Burges and A. Smola, MIT-Press.*

[8] Joachims, T. (2008), Program Codes for SVM-Light and SVM-Multiclass. *Available at http:svmlight.joachims.org/ .*

[9] Ji?ina, Marcel (2003), Classification of Multivariate Data Using Distribution Mapping Exponent. *Proceedings of the International Conference in Memoriam John von Neumann* Budapest : Budapest Muszaki Foiskola (Budapest Polytechnic), 2003 - (Szakl, A.), s. 155-168 ISBN 963-7154-21-3. [International Conference in Memorial John von Neumann. Budapest (HU), 12.12.2003]

[10] Jiřina, M. and Jiřina, Jr., M. (2013) Fractal Based Data Separation in Data Mining. *Proceedings of the The Third International Conference on Digital Information Processing and Communications* Hong Kong : SDIWC, 2013, pp. 287-295. ISBN 978-0-9853483-3-5. [ICDIPC 2013 - International Conference on Digital Information Processing and Communications /3./. Dubai (AE), 30.01.2013-01.02.2013]

[11] Jiřina, M. and Jiřina, Jr., M. (2013), Utilization of Singularity Exponent in Nearest Neighbor Based Classifier. *Journal of Classification (Springer)*, Vol. 30, No. 1, pp. 3-29. ISSN 0176-4268.

[12] Jiřina, M. and Jiřina, Jr., M. (2014), Correlation Dimension Based Classifier. *IEEE Transactions on Cybernetics*, Vol. 44, pp. 2253-2263. ISSN 2168-2267.

[13] Jiřina, M. and Jiřina, Jr., M. (2015), Classification Using Zipfian Kernel. *Journal of Classification (Springer)*, Vol. 32, No. 2, pp. 305-326. ISSN 0176-4268.

[14] Lichman, M. (2013), UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. *online*,http:archive.ics.uci.edu/ml

[15] Oneto, L., Ridella, S., Anguita, D.(2016), Tikhonov, Ivasnov and Morozov regularization for support vector machine learning. *Machine Lerning (Springer)*, Vol. 2016, pp. 103-136.

[16] Paredes, R. (2008), CPW: Class and Prototype Weights learning. *[online]*, Available: http:www.dsic.upv.es/ rparedes/research/CPW/index.html.

[17] Paredes. R. (2010), Data sets corpora. *[online]*, Available: http:algoval.essex.ac.uk/data/vector/UCI/, in fact, the primary source is S. M. Lucas, Algoval: Algorithm Evaluation over the Web.

[18] Paredes, R., Vidal, E. (2006), Learning Weighted Metrics to Minimize Nearest Neighbor Classification Error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 7, pp. 1100-1110, (July 2006).