národní
úložiště
šedé
literatury

## Testing Random Forests for Unix and Windows

Jiřina, Marcel
2010
Dostupný z http://www.nusl.cz/ntk/nusl-41904

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

Tento dokument byl stažen z Národního úložišť̌ šedé literatury (NUŠL).
Datum stažení: 11.04.2024
Další dokumenty můžete najít prostřednictvím vyhledávacího rozhraní nusl.cz .

Institute of Computer Science
Academy of Sciences of the Czech Republic

# Testing Random Forests for Unix and Windows 

## Marcel Jiřina and Marcel Jiřina, jr.

## Technical Report No. V-1075

July 2010


#### Abstract

The Random Forest is a method and also a program for data clustering and classification. Especially in classification the Random Forest method appears to be the best approach perhaps among all others up to now. The message of this report is a new version of famous RandForest program written by Leo Breiman and Adele Cutler [3]. In the original version written in Fortran 77 all information about data to be processed and details of processing must be included in the program source text. In our modification in Fortran 90 there are binaries for Windows and Linux and information for different tasks is passed with help of arguments. After a brief description of our modification of RandForest the detailed manual follows. The next Chapter describes testing data corpora used and gives results of the Random Forest program. The program source text and binaries, and results are free under GNU General Public License.


Keywords:
Random forest, RandFor, multivariate data, Clustering, classification, data mining, probability estimation.

## Contents

1 Introduction ..... 3
2 Command line Random Forest program ..... 3
2.1 Motivation. ..... 3
2.2 Terminology ..... 3
2.3 Important features of FORTRAN ..... 4
2.4 Resulting changes of the Random Forests program ..... 4
2.5 Adds-on ..... 5
2.5.1 Register.txt file ..... 5
2.5.2 ROC.txt file ..... 5
3 The Manual ..... 5
4 Testing ..... 7
4.1 Data set corpora - tasks from UCI Machine Learning Repository ..... 7
4.2 Results ..... 8
5 Conclusion ..... 9
References ..... 10
6 Appendix 1 - A part of the original manual (with some corrections) ..... 11
6.1 Random forests computes ..... 11
6.2 Setting Parameters ..... 11
6.2.1 Line 1 Describing The Data ..... 11
6.2.2 Line 2 Setting up the run ..... 12
6.2.3 Line 3 Options on Variable Importance ..... 13
6.2.4 Line 4 Options based on proximities ..... 13
6.2.5 Line 5 Transform to Principal Coordinates ..... 14
6.2.6 Line 6 Saving the forest ..... 14
6.2.7 Line 7 Output Controls ..... 14
6.2.8 USER WORK ..... 14
6.2.9 REMARKS ..... 15
6.3 Outline of How Random Forests Works ..... 16
6.3.1 Usual Tree Construction--Cart ..... 16
6.3.2 Random Forests Construction. ..... 16
6.3.3 Random Forests Tools ..... 16
6.3.4 Test Set Error Rate ..... 16
6.3.5 Class probability estimates ..... 17
6.3.6 Variable Importance ..... 17
7 Appendix 2 - The progran listing with comments (shortened) ..... 18

## Testing Random Forest for Unix and Windows

Marcel Jiřina and Marcel Jiřina, Jr. (marcel@cs.cas.cz)

## 1 Introduction

The Random Forest is a method and also a program for mdata mining and pattern rrecognition. It comprises clustering and classification by learning. Especially in classification the Random Forest method appears the best approach perhaps among all others up to now. Note that „Random Forests(tm) is a trademark of Leo Breiman and Adele Cutler and is licensed exclusively to Salford Systems for the commercial release of the software. Our trademarks also include RF(tm), RandomForests(tm), RandomForest(tm) and Random Forest(tm)", see [11].
The message of this report is a new version of famous Random Forests program "rf5new0.for" written by Leo Breiman and Adele Cutler [3] in Fortran 77. In the original version all information about data to be processed and details of processing must be included in the program source text. Thus modified program must be compiled and afterwards run with data. In our modification the program is rewritten into Fortran 90 and a user need not compile the program. Information for different tasks is passed to program with help of arguments. The program has one source text and binaries can run under UNIX/LINUX or under Windows environment. A brief description of our modification of Random Forest is given first and then the detailed manual follows. The next Chapter describes testing data corpora used and gives results of processing by the Random Forests program. In the Appendix there is a verbatim copy of a part of original manual with some our corrections included. The program and results are free under GNU General Public License [6]. (In short, it is free for non-profit use, there is no warranty, and the citation of original source [1] and of this report is mandatory when published in any form, see below ${ }^{1}$.)

## 2 Command line Random Forest program

### 2.1 Motivation

We found the original Random Forests program rather difficult to use. The necessity to set up (change) parameters in the source text of the FORTRAN program is rather effective for tuning and experimentation. For user who has no knowledge about Fortran 77 programming language and about peculiarities of the use of a compiler, it may be a stressful task. This feeling may result in the use another classifier that is friendlier to use.

### 2.2 Terminology

The task computed has different parameters, e.g. the name of the training data file, the dimensionality of the task, the number of classes when task is the classification problem, and so like. The name of the program and parameters form a command line. When speaking about command line, the first item in it is the name of program optionally with path. The items following the program's name are called arguments. In fact, arguments are the parameters in the sense above. The meaning of arguments is given by their order or there are named

[^0]arguments. The named argument consists of a name and of a value of parameter. Between the name and the value some separator can be found, usualy a colon or equivalence sign.

### 2.3 Important features of FORTRAN

In Fortran 77 a feature of dynamic allocation does not exist. On the other hand, there is a feature in Fortran 77 as well as in Fortran 90 that one can easily redefine dimensions of an array, i.e. change the number of dimensions and their size in the subroutine or function. This allows moving large part of the main progran into subroutine without changing any declaration of arrays and variables. The cost if it is that names of all arrays and of all variables shared with the main program must appear as formal parameters of such a subroutine. To formal parameters of a subroutine must correspond true parameters. And this was the main part of our effort to make the Random Forests program friendlier in the sense that it can be used as easily as any other program run from the command line. To the same source program there are run-time binaries for running under Windows or UNIX/Linux environment.

### 2.4 Resulting changes of the Random Forests program

The change from Fortran 77 to Fortran 90 can be viewed as a formal task concerning comments (the comment line starts with letter c in Fortran 77 and with "!" in Fortran 90) and continuation lines (instead of a new line starting with \& or another character in column 6, the \& character must be the last character of the preceding line in Fortran 90).

The most important was the change of all FORTRAN parameters (the statement "parameter (...)") into variables. For it the declaration of parameters as variables has to be separated from setting up their default values (all declarations must precede the first executable statement in FORTRAN). To enable user to change parameter's value the analysis of the command line attributes has been included.

To allow class marks in form of a text, the analysis of class marks in the learning file and a tranformation of them into numbers $1,2 \ldots$ was included at the beginning of the program. In a corresponding way the procedures for reading data.train and data.test files has been modified accordingly.

A large part of the main program has been transformed into subroutine MAIN2 without changing any declaration of arrays and variables. The names of all arrays and of all variables shared with the main program must appear as formal parameters of such a subroutine. To formal parameters of a subroutine must correspond some true parameters. To construct true parameters for arrays three vast integer, real, and double precision arrays has been declared and necessary amount of memory assigned to them dynamically according to size of the task solved. In Fortran 90 (not in Fortran 77 ) there exist a declaration ALLOCTABLE and executable statement ALLOCATE for this purpose. The individual arrays in MAIN2 subroutine correspond to some part of one of these large arrays of the same type (integer, real or double precision) and form true parameters dynamically at the time when subroutine MAIN2 is called.

Parameters for print and for storing data in files have been set up to minimize reporting. At the beginning of run some input information is printed. Then some dots appear to show that
program is not "dead" during long run with large data sets. Finally resulting classification errors appears.

Two additional files "register.txt" and "ROC.txt" can be generated.

### 2.5 Adds-on

### 2.5.1 Register.txt file

In the end of run a line is written to the file "register.txt". If no such file exists, it is generated else a new line is appended to existing file. The line consists of names of training and testing files, the dimensionality of the task, the number of classes, the learning classification error, and the testing classification error.

### 2.5.2 ROC.txt file

If parameter ROC is set up to 1 (i.e. argument $R O C=1$ appears in the command line) then a matrix of class probabilities together with the true class number and by the program found class number appear as a file "ROC.txt". In more detail - in the first line the names of the training and testing files appear, the second line gives headings for data on next lines. The third and other lines of this file give the true class number, by the program found class number, and individual class probabilities. Each line corresponds to one sample of the testing set (test file) in the same order as in the testing set. When there is a two class problem then information in ROC.txt file can be used for constructing the famous ROC curve [ 7]. Note that existing ROC.txt file is rewritten during the run of the program by the new one without warning.

## 3 The Manual

This part is, in fact, the help screen shown (see box below) when the program is run with less than three arguments. The following three arguments are mandatory in the fixed order:

The first argument is the file name of the training set, optionally with a path.
The second argument is the file name of the testing set, optionally with a path.
The third argument $n$ is the dimensionality of the task.
When there is no testing set or no training set a dummy argument is used.
Data is supposed to be organized in rows, items on the row separated by tabulator character or by one or more spaces. Each row represents one sample, pattern, event or object. There are $n+1$ items on the row. The first $n$ items must be numeric; the last item means a class to which the sample belongs. The class mark need not be numeric. Alfabetic or alfanumeric class marks are also admissible. When numeric class marks are used they need not form an uniterrupted series $1,2 \ldots$ Our program recognizes individual classes and their total number automatically. Also the numbers of samples (rows) in the training and testing set are recognized automatically.

The other arguments in the command line can be the named arguments. The equivalence sign is used as a separator between parameter's name and its value. All named arguments are optional. When not stated explicitly in the command line the default values are used.

```
RandomForest modified by MJ&MJ,jr. during May and June 2010
    Fortran 90 version with dynamic memory allocation.
    Free under GNU General Public License.
    When published please cite as:
    -Breiman,L.: Random Forests, Machine Learning Vol. 45, No. 1, pp. 5-32.(2001)
    -Jirina,M., Jirina, M., jr.: Testing Random Forest for Unix and Windows.
        Technical Report No. V-1075, Institue of Informatics AS CR, Prague,
        Czech Republic (2010).
Usage: prog.exe datatrain datatest mdim [named parameters]
DESCRIBE DATA (with optional defaults)
Data files: numeric with class labels as integers, reals, or
    strings.
datatrain: training data file name
datatest: testing data file name
mdim: task dimensionality (number of variables)]
Named parameters: Form: Parname=Parvalue.
[ntrain: number of samples (cases) in the training data - NOT NECESSARY NOW!]
[nclass: number of classes - NOT NECESSARY NOW!]
[ntest: the number of samples (cases) in the test set. NOTE: Put
        ntest=1 if there is no test set. Putting ntest=0 may cause
        compiler complaints.
        - NOT NECESSARY NOW if data test file exists!]
maxcat: the largest number of values assumed by a categorical
    variable in data
labeltr=1 if the data has class labels. If not, =1 or 2 adds a
    synthetic class
labelts=0 if the test set has no class labels, 1 if the test set
    has cl.labels
SET RUN PARAMETERS
mtry0=number of variables randomly selected at each node
    Default=int(sqrt(float(mdim))+0.5), (originally 2 later on 5).
    Needs tuning! Begin with this value and try a value twice as high
    and half as low. See manu
ndsize=1=minimum node size
jbt=500=number of trees to grow
look=100=how often you want to check the prediction error
lookcls=1=show on the screen or not
jclasswt=0
mdim2nd=0
mselect=0
OUTPUT CONTROLS; all defaults zero:
ROC=1 to get data for ROC curve i.e. outputs for each test sample
    else 0. (file roc.txt); default 0.
isumout = 0/1 1=summary to screen. Includes err rates & confusion
    matrix
idataout = 0/1/2 1=train,2=adds test (7)
impfastout = 0/1 1=gini fastimp(8)
impout = 0/1/2 1=imp,2=to screen(9)
impnout = 0/1 1=impn (10)
interout=0/1/2 1=interaction, 2=screen (11)
iprotout=0/1/2 1=prototypes,2=screen (12)
iproxout=0/1/2 1=prox,2=adds test (13)
iscaleout = 0/1 1=scaling coors(14)
ioutlierout=0/1/2 1=train,2=adds test (15)
NAME OUTPUT FILES FOR SAVING THE FOREST STRUCTURE
isaverf=1 savedforest
```

```
isavepar=1 savedparams
isavefill=1 savedmissfill
isaveprox=1 savedprox
NAME OUTPUT FILES TO SAVE DATA FROM CURRENT RUN
idataout=1 save-data-from-run
impfastout=1 save-impfast
impout=1 save-importance-data
impnout=1 save-caseimp-data
interout=1 save-pairwise-effects
iprotout=1 save-protos
iproxout>=1 save-run-proximities
iscaleout=1 save-scale
ioutlierout>=1 save-outliers
iviz=1 the graphics program is to be used.
```

Box 1. The help screen printed when the program is run with less than three arguments.

## 4 Testing

The testing should show the classification ability of the method for some tasks and also shows the classification ability relative to the other published methods and the results for the same data sets. Here we do not compare results obtained with any published result gained with other methods. Our task is to present reproducible results and any comparisons are up to a kind reader.

### 4.1 Data set corpora - tasks from UCI Machine Learning Repository

We used real-life tasks from the UCI Machine Learning Repository; see Asuncion and Newman [8]. 24 databases have been used for the classification task into two to 26 classes. The number of attributes not including the class mark differs from 4 to 180 . Basic characteristics of data sets are summarized in Table 1. Data originally from the UCI Machine learning repository [8] were gained mostly from R. Paredes [5] (denoted by P in column Source in the table). These data sets are ready for a run with a classifier. We used all data sets in this corpus. Each task consists of 50 pairs of training and testing sets corresponding to 50fold cross validation. For DNA data [5], Letter data (Letter recognition [8]), and Satimage (Statlog Landsat Satellite [8]) the single partition into training and testing sets according to specification in [8] was used. We also added the popular Iris data set. Iris data were taken from [8] but we use them without Setoza class, i.e. we used two classes Versicolor and Virginica only according to Friedman [9] and then we have split remaining data into 10 pairs for ten-fold cross validation.

| Dataset | Dimension <br> (\# <br> attributes) | Number <br> of classes | Total <br> samples | Learning <br> set size | Test set <br> size | Cross <br> 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 |
| DNA | 180 | 3 | 31186 | 2000 | 1186 | 1 | P 2 |


| 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 |

Table 1. Characteristics of data sets basically from the UCI Machine learning repository gained from or modified according to different sources. Abbreviations for sources: PParedes [10]; P2 - Paredes [5]; UCI MLR - Asuncion and Newman [8]. Note (1): Iris data are used without Setoza class, i.e. two classes Versicolor and Virginica only according to Friedman [9].

### 4.2 Results

The classification errors for data sets mentioned above are given in Table 2. Errors were computed for two values of mtry 0 . (mtry0 is the number of variables randomly selected at each node). One value follows from the recommendation to use the number of variables approximately equal to the square root of the dimensionality $n$ (mdim) of the task. The other value is the default value (2) found in the original program text of the RandFor program. It is seen that differences are small and mostly in an advantage of the former option.

| Data <br> set | $\mathrm{mtry} 0=$ <br> int $(\mathrm{sqrt}(n+0.5)$ | $\mathrm{mtry} 0=$ <br> 2 |
| :--- | :---: | :---: |
| australian | 0.1276 | 0.1277 |
| balance | 0.1838 | 0.1972 |
| Cancer | 0.0299 | 0.0302 |
| diabetes | 0.2328 | 0.2322 |
| DNA | 0.0540 | 0.0565 |
| german | 0.2386 | 0.2376 |
| Glass | 0.2387 | 0.2354 |
| Heart | 0.1811 | 0.1822 |
| ionosphere | 0.0644 | 0.0658 |
| Iris | 0.0491 | 0.0591 |


| led17 | 0.0000 | 0.0000 |
| :--- | :--- | :--- |
| Letter | 0.0507 | 0.0373 |
| Liver | 0.2881 | 0.2884 |
| monkey1 | 0.0152 | 0.0098 |
| phoneme | 0.1198 | 0.1184 |
| satimage | 0.0900 | 0.0880 |
| segmen | 0.0263 | 0.0252 |
| Sonar | 0.2302 | 0.2216 |
| Vehicle | 0.2525 | 0.2501 |
| Vote | 0.0347 | 0.0343 |
| Vowel | 0.0389 | 0.0401 |
| waveform21 | 0.1468 | 0.1469 |
| waveform40 | 0.1447 | 0.1434 |
| Wine | 0.0250 | 0.0244 |

Table 2. Summary of classification errors for 24 tasks from the UCI Machine Learning Repository according to Table 1 and two settings of the value of the mtry0, i.e. the number of variables randomly selected at each node.

## 5 Conclusion

We found the original Random Forest program rather difficult to use. The necessity to set up (change) parameters in the source text of the FORTRAN program is rather effective for tuning and experimentation. For user who has no knowledge about Fortran 77 programming language and about peculiarities of the use of a compiler, it may be a stressful task. This feeling may result in the use of another classifier that is friendlier to use.

To use a dymnamic memory allocation we had to move from Fortran 77 to Fortran 90. There is a feature of the FORTRAN language (' 77 as well as ' 90 ) that one can easily redefine dimensions of an array, i.e. change the number of dimensions and their size in the subroutine or function. This allows moving large part of the main progran into subroutine without changing any declaration of arrays and variables. The cost if it is that names of all arrays and of all variables shared with the main program must appear as formal parameters of such a subroutine. Some true parameters must correspond to formal parameters of a subroutine. And this was the main part of our effort to make the Random Forest program friendlier in the sense that it can be used as easily as any other program run from the command line. To the same source program in Fortran 90 there are run-time versions of binaries for Windows or UNIX/Linux environment.

## Acknowledgements

This work was supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M0567, and No. MSM6840770012
Transdisciplinary Research in the Field of Biomedical Engineering II. Authors are also indebted to professor M. Said Krayem, University of Aleppo, Aleppo, Syria for his valuable notes and discussions about problems solved in this report..

## References

[1]Breiman, L.: Random Forests, Machine Learning Vol. 45, No. 1, pp. 5-32. (2001)
[2] Breiman, L.: Manual On Setting Up, Using, And Understanding Random Forests V3.1, On-line http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf (2002), read 1 June 2010)
[3] Leo Breiman and Adele Cutler: The original Fortran code of the RandFor. On-line http://oz.berkeley.edu/users/breiman/RandomForests/cc_home.htm (read 26.5.2010).
[4] S. M. Lucas, Algoval: Algorithm Evaluation over the Web, [online], 2008, [cited November 23, 2008]. Available: [http://algoval.essex.ac.uk/data/vector/UCI/](http://algoval.essex.ac.uk/data/vector/UCI/).
[5] R. Paredes: CPW: Class and Prototype Weights learning, [online], 2008, [cited November 23, 2008]. Available: [http://www.dsic.upv.es/~rparedes/research/CPW/index.html](http://www.dsic.upv.es/~rparedes/research/CPW/index.html). [6] GNU General Public License. On-line http://www.gnu.org/licenses/gpl.html (read 14 June 2010)
[7] Receiver operating characteristic. On-line http://en.wikipedia.org/wiki/Receiver_operating _characteristic (read 14 June 2010)
[8] A. Asuncion, D.J. Newman: UCI Machine Learning Repository
[http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science. (Read 14 June 2010)
[9] Friedmann, J. H.: Flexible Metric Nearest Neighbor Classification. Technical Report 113, Dept. of Statistics, Stanford University, 1994.
[10] Paredes. R.: Data sets corpora. [online] http://algoval.essex.ac.uk/data/vector/UCI/ (Read 14 June 2010), in fact, the primary source is S. M. Lucas, Algoval: Algorithm Evaluation over the Web, [online]. Available: http://algoval.essex.ac.uk/data/vector/UCI/ [cited 14 June 2010] [11] http://oz.berkeley.edu/users/breiman/RandomForests/cc_home.htm [On-line], (Read 26.5.2010)

## 6 Appendix 1 - A part of the original manual (with some corrections)

Here we give Chapters 1 till 9.3 from the original manual [2]. Some places are slightly changed to fit better to the original RandFor program as well as to our command line controlled program.

## Manual On Setting Up, Using, And Understanding Random Forests V3.1

The V3.1 version of random forests contains some modifications and major additions to Version 3.0. It fixes a bad bug in V3.0. It allows the user to save the trees in the forest and run other data sets through this forest. It also allows the user to save parameters and comments about the run.
I apologize in advance for all bugs and would like to hear about them. To find out how this program works, read my paper "Random Forests" Its available on the same web page as this manual. It was recently published in the Machine Learning. Journal The program is written in extended Fortran 77 making use of a number of VAX extensions. It runs on SUN workstations $f 77$ and on Absoft Fortran 77 (available for Windows) and on the free g77 compiler. but may have hang ups on other f77 compilers. If you find such problems and fixes for them, please let me know.

### 6.1 Random forests computes

- classification and class probabilities
- intrinsic test set error computation
- principal coordinates to use as variables.
- variable importance (in a number of ways)
- proximity measures between cases
- a measure of outlyingness
- scaling displays for the data

The last three can be done for the unsupervised case i.e. no class labels. I have used proximities to cluster data and they seem to do a reasonable job. The new addition uses the proximities to do metric scaling of the data. The resulting pictures of the data are interesting and useful.
The first part of this manual contains instructions on how to set up a run of random forests V3.1. The second part contains the notes on the features of random forests V3.1 and how they work.

### 6.2 Setting Parameters

The first seven lines following the parameter statement need to be filled in by the user.

### 6.2.1 Line 1 Describing The Data

mdim=number of variables
nsample $\mathbf{0}=$ number of cases (examples or instances) in the data nclass $=$ number of classes
maxcat=the largest number of values assumed by a categorical variable in the data
ntest=the number of cases in the test set. NOTE: Put ntest=1 if there is no test set. Putting ntest $=0$ may cause compiler
complaints.
labelts $=0$ if the test set has no class labels, 1 if the test set has class
labels.
iaddel $=\mathbf{0}$ labeltr $=\mathbf{1}$ if the train data has class labels. If not, iaddcl=1 or 2
adds a synthetic class as described below
If their are no categorical variables in the data set maxcat=1. If
there are categorical variables, the number of categories assumed by each categorical variable has to be specified in an integer vector called cat, i.e. setting cat(5)=7 implies that the 5th variable is a categorical with 7 values. If maxcat $=1$, the values of cat are automatically set equal to one. If not, the user must fill in the values of cat in the early lines of code.
For a J-class problem, random forests expects the classes to be numbered $1,2, \ldots, \mathrm{~J}$. For an L valued categorical, it expects the values to be numbered $1,2, \ldots, \mathrm{~L}$. At present, L must be less than or equal to 32 .
A test set can have two purposes--first: to check the accuracy of RF on a test set. The error rate given by the internal estimate will be very close to the test set error unless the test set is drawn from a different distribution. Second: to get predicted classes for a set of data with unknown class labels. In both cases the test set must have the same format as the training set. If there is no class label for the test set, assign each case in the test set label classs \#1, i.e. put $\operatorname{cl}(\mathrm{n})=1$, and set labelts $=0$. Else set labelts $=1$.
If the data has no class labels, addition of a synthetic class enables it it to be treated as a two-class problem with nclass $=2$. Setting iaddclass $=1$ forms the synthetic class by independent sampling from each of the univariate distributions of the variables in the original data. Setting iaddclass $=2$ forms the synthetic class by independent sampling from uniforms such that each uniform has range equal to the range of the corresponding variable.

### 6.2.2 Line 2 Setting up the run

$\mathbf{j b t}=$ number of trees to grow. Default value is originally 500 . If error message „segmentation fault" appears then use a smaller value.
This is the number of trees to be grown in the run. Don't be stingy--random forests produces trees very rapidly, and it does not hurt to put in a large number of trees. If you want auxiliary information like variable importance or proximities grow a lot of trees--say a 1000 or more. Sometimes, I run out to 5000 trees if there are many variables and I want the variables importances to be stable.
$\mathbf{m t r y 0}$ [corrected, zero added] = number of variables randomly selected at each node This is the only parameter that requires some judgment to set, but forests isn't too sensitive to its value as long as it's in the right ball park. I have found that setting mtry0 equal to the square root of mdim gives generally near optimum results2. My advice is to begin with this value and try a value twice as high and half as low monitoring the results by setting look=1 and checking the internal test set error for a small number of trees. With many noise variables present, mtry0 has to be set higher.
look=how often you want to check the prediction error random forests carries along an internal estimate of the test set error as the trees are being grown. This estimate is outputted to

[^1]the screen every look trees. Setting look=10, for example, gives the internal error output every tenth tree added. If there is a labeled test set, it also gives the test set error. Setting look=jbt+1 eliminates the output. Do not be dismayed to see the error rates fluttering around slightly as more trees are added. Their behavior is analagous to the sequence of averages of the number of heads in tossing a coin.
$\mathbf{i p i}=$ set priors. pi is an real-valued vector of length nclass which sets prior
probabilities for classes. ipi=0 sets these priors equal to the class
proportions. If the class proportions are very unbalanced, you may
want to put larger priors on the smaller classes. If different
weightings are desired, set $\mathrm{ipi}=1$ and specify the values of the $\{\mathrm{pi}(\mathrm{j})\}$
early in the code. These values are later normalized, so setting $\mathrm{pi}(1)=1, \mathrm{pi}(2)=2$ weights a class 2 instance twice as much as a class 1 instance. The error rates reported are an unweighted count of misclassified instances.
ndsize=minimum node size; setting this to the value k means that no node with fewer than k cases will be split. The default that always gives good performances is ndsize $=1$. In large data sets, memory requirements will be less and speed enchanced if ndsize is set larger. Usually, this results in only a small loss of accuracy for large data sets.

### 6.2.3 Line 3 Options on Variable Importance

imp=1 turns on the variable importances methods described below.
impstd=1 gives the standard imp output
impmargin $=1$ gives, for each case, a measure of the effect of noising up each variable
impgraph=1 gives for each variable, a plot of the effect of the variable on the class probabilities.
impstd $=1$ computes and prints the following columns to a file
i) variable number
variables importances computed as:
ii) The $\%$ rise in error over the baseline error.
iii) $100 *$ the change in the margins averaged over all cases
iv) The proportion of cases for which the margin is decreased minus the proportion of increases.
v) The gini increase by variable for the run
impgraph $=1$ computes and prints out the columns for each
variable m--
i) variable number i.e. $m$
ii) sorted values of $x(m)$ from lowest to highest
iii-iii+nclass) effect of $x(m)$ on the probabilities of class $j$.

### 6.2.4 Line 4 Options based on proximities

iprox=1 turns on the computation of the intrinsic proximity measures between any two cases. This has to be turned on for the following options to work.
noutlier $=1$ computes an outlyingness measure for all cases in the data. If iaddcl=1 then the outlyingness measure is computed only for the original data. The output has the columns :
i) class
ii) case number
iii) measure of outlyingness
iscale $=1$ computes scaling coordinates based on the proximity
matrix. If iaddcl is turned on, then the scaling is outputted only for
the original data. The output has the columns:
i) case number
i) true class
iii) predicted class.
iv) 0 if ii)=iii), 1 otherwise
$\mathrm{v}-\mathrm{v}+\mathrm{msdim}$ ) scaling coordinates
mdimsc is the number of scaling coordinates to be extracted.
Usually 4-5 is sufficient

### 6.2.5 Line 5 Transform to Principal Coordinates

$\mathbf{i p c}=1$ takes the x -values and computes principal coordinates from the covariance matrix of the x's. These will be the new variables for RF to operate on. This will not work right if some of the variables are categorical.
mdimpc: This is the number of principal components to extract. It has to be $<=$ mdim.
norm $=1$ normaizes all of the variables to mean zero and sd one before computing the principal components.

### 6.2.6 Line 6 Saving the forest

isavef $=1$ saves all the trees in the forest to a file named eg. A.
isavep $=1$ creates a file B that contains the parameters used
in the run and allows up to 500 characters of text description about the run.
irunf=1 reads file $A$ and runs new data down the forest.
ishowp $=1$ reads file $B$ and prints it to the sccreen
The calling code and files names required (except for the name of A) are at the end of the main program. The name for $A$ is entered at the beginning of the program.

### 6.2.7 Line 7 Output Controls

Note: user must supply file names for all output listed below or send it to the screen.
nsumout=1 writes out summary data to the screen. This includes errors rates and the confusion matrix
infout $=1$ prints the following columns to a file
i) case number
ii) 1 if predicted class differs from true class, 0 else
iii) true class label
iv) predicted class label
v) margin=true class prob. minus the max of the other class prob.
vi)-vi+nclass) class probabilities
ntestout=1 prints the follwing coumns to a file
i) case number in test set
ii) true class (true class $=1$ if data is unlabeled)
iii) predicted class
iv-iv+nclass) class probabilities
iproxout=1 prints to file
i) case \#1 number
ii) case \#2 number
iii) proximity between case \#1 and case \#2

### 6.2.8 USER WORK

The user has to construct the read-in the data code of which I have left an example. This needs to be done after the dimensioning of
arrays. If maxcat $>1$ then the categorical values need to be filled in. If ipi=1, the user needs to specify the relative weights of the classes. File names need to be specified for all output. This is important since a chilling message after a long run is "file not specified" or something similar.

### 6.2.9 REMARKS

The proximities can be used in the clustering program of your choice. Their advantage is that they are intrinsic rather than an ad hoc measure. I have used them in some standard and home-brew clustering programs and gotten reasonable results. The proximities between class 1 cases in the unsupervised situation can be used to cluster. Extracting the scaling coordinates from the proximities and plotting scaling coordinate $i$ versus scaling coordinate $j$ gives illuminating pictures of the data. Usually, $\mathrm{i}=1$ and $\mathrm{j}=2$ give the most information (see the notes below).
There are four measures of variable importance: They complement each other. Except for the 4th they are based on the test sets left out on each tree construction. On a microarray data with 5000 variables and less than 100 cases, the different measures single out much the same variables (see notes below). But I have found one synthetic data set where the 3rd measure was more sensitive than the first three.
Sometimes, finding the effective variables requires some hunting. If the effective vzriables are clear-cut, then the first measure will find them. But if the number of variables is large compared to the number of cases, and if the predictive power of the individual variables is small, the other measures can be useful.
Random forests does not overfit. You can run as many trees as you want. Also, it is fast. Running on a 250 mhz machine, the current version using a training set with 800 cases, 8 variables, and mtry $=1$, constructs each tree in .1 seconds. On a training set with 2200 cases, 11 variables, and mtry $=3$, each tree is constructed in .2 seconds. It takes 4 seconds per tree on a training set with 15000 cases and 16 variables with mtry=4, while also making computations for a 5000 member test set.
The present version of random forests does not handle missing values. A future version will. It is up to the user to decided how to deal with these. My current preferred method is to replace each missing value by the median of its column and each missing categorical by the most frequent value in that categorical. My impression is that because of the randomness and the many trees grown, filling in missing values with a sensible values does not effect accuracy much.
For large data sets, if proximities are not required, the major memory requirement is the storage of the data itself, and the three integer arrays a,at,b. If there are less than 64,000 cases, these latter three may be declared integer*2 (non-negative). Then the total storage requirement is about three times the size of the data set. If proximities are calculated, storage requirements go up by the square of the number of cases times eight bytes (double precision).

### 6.3 Outline of How Random Forests Works

### 6.3.1 Usual Tree Construction--Cart

Node=subset of data. The root node contains all data.
At each node, search through all variables to find best split into two children nodes.
Split all the way down and then prune tree up to
get minimal test set error.

### 6.3.2 Random Forests Construction

Root node contains a bootstrap sample of data of same size as original data. A different bootstrap sample for each tree to be grown.
An integer K is fixed, $\mathrm{K} \ll$ number of variables. K is the only parameter that needs to be specified. Default is the square root of number of variables.
At each node, K of the variables are selected at random. Only these variables are searched through for the best split. The largest tree possible is grown and is not pruned.
The forest consists of N trees. To classify a new object having coordinates x , put x down each of the N trees. Each tree gives a classification for x .
The forest chooses that classification having the most out of N votes.
Transformation to Principal Coordinates
One of the users lent us a data set in which the use of a few principal components as variables reduced the error rate by $2 / 3$ rds. On experimenting, a few other data sets were found where the error rate was significantly reduced by pre-transforming to principal coordinates As a convenience to users, a pretransformation subroutine was incorporated into this version.

### 6.3.3 Random Forests Tools

The design of random forests is to give the user a good deal of information about the data besides an accurate prediction. Much of this information comes from using the "out-of-bag" cases in the training set that have been left out of the bootstrapped training set.
The information includes:
a) Test set error rate.
b) Variable importance measures
c) Intrinsic proximities between cases
d) Scaling coordinates based on the proximities
e) Outlier detection

The following explains how these work and give applications, both for labeled and unlabeled data.

### 6.3.4 Test Set Error Rate

In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is gotten internally, during the run, as follows:
Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree. Test Set Error Rate

Put each case left out in the construction of the kth tree down the kth tree to get a classification.
In this way, a test set classification is gotten for each case in about one-third of the trees. Let the final test set classification of the forest be the class having the most votes.
Comparing this classification with the class label present in the data gives an estimate of the test set error.

### 6.3.5 Class probability estimates

At run's end, for each case, the proportion of votes for each class is recorded. For each member of a test set (with or without class labels), these proportions are also computed. By a stretch of terminology, we call these class probability estimates. These should not be interpreted as the underlying distributional probabilities. But they contain useful information about the case.
The margin of a case is the proportion of votes for the true class minus the maximum proportion of votes for the other classes. The size of the margin gives a measure of how confident the classification is.

### 6.3.6 Variable Importance

Because of the need to know which variables are important in the classification, random forests has four different ways of looking at variable importance. Sometimes influential variables are hard to spot--using these four measures provides more information.
Measure 1
To estimated the importance of the mth variable. In the left out cases for the kth tree, randomly permute all values of the mth variable Put these new covariate values down the tree and get classifications.
Proceed as though computing a new internal error rate. The amount by which this new error exceeds the original test set error is defined as the importance of the mth variable.
Measures 2 and 3
For the nth case in the data, its margin at the end of a run is the proportion of votes for its true class minus the maximum of the proportion of votes for each of the other classes. The 2nd measure of importance of the mth variable is the average lowering of the margin across all cases when the mth variable is randomly permuted as in method 1.
The third measure is the count of how many margins are lowered minus the number of margins raised.

## Measure 4

The splitting criterion used in RF is the gini criterion--also used in CART. At every split one of the mtry variables is used to form the split and there is a resulting decrease in the gini. The sum of all decreases in the forest due to a given variable, normalized by the number of trees, froms measure 4.
We illustrate the use of this information by some examples. Some of these were done on version 1 so may differ somewhat from the version 3 output.
[The remaining part with examples is omitted here.]

## 7 Appendix 2 - The progran listing with comments (shortened)

```
    program rf5new
    MODIFIED by MJ&MJ,jr, marcel@cs.cas.cz
        May, June 2010. Important changes when transfered into Fortran 90
        made inside can be found by searching for '2010'.
        Suitable for numerical data and class labels as strings.
        Classes need not be numbered 1,2,...nclass (=NoOfClasses).
    Representation of the class labels as strings, integers or reals
    is admissible.
    [My file numbers are 33 for register.txt and 34 for reading train
    and test data files in the "main2".]
    ORIGINAL
    Copyright 2002-2003 Leo Breiman and Adele Cutler
    This is free open source software but its use,in part or
    in whole,in any commercial product that is sold for profit
    is prohibited without the written consent of Leo Breiman
    and Adele Cutler.
    We very much appreciate bug notices and suggested improvements.
    leo@stat.berkeley.edu adele@math.usu.edu
    SET ALL PARAMETERS FIRST GROUP BELOW. GENERALLY,
    SETTING PARAMETERS TO ZERO TURNS THE CORRESPONDING
    OPTION OFF.
    ALL RELEVANT OUTPUT FILES MUST BE GIVEN NAMES--SEE BELOW.
    WORKING VARIABLES DECLARATIONS
    character msg1*39, msg2*39, txt*39,ParName*39,parstr*39,pb*39,chr
    character*25 stri(100),stri1,stri2 !27.5.2010
    real xx
    ATTRIBUTES DECLARATIONS
    character*255 datatrain, datatest
    integer mdim,ntrain,nclass,maxcat,ntest,&
    labelts,labeltr,mtry0, ndsize, jbt,look,lookcls,&
    jclasswt,mdim2nd,mselect,imp,interact,impn,&
    nprox,nrnn,noutlier,nscale,nprot,missfill,iviz,&
    isaverf,isavepar,isavefill,isaveprox,&
    irunrf,ireadpar,ireadfill,ireadprox,&
    isumout,idataout,impfastout,impout,impnout,interout,&
    iprotout,iproxout,iscaleout,ioutlierout
    integer nsample,nrnodes,mimp,near,&
    ifprot,ifscale,iftest,mdim0,ntest0, nprot0, nscale0,&
    ROC
!
    integer Sizereal, Sizeint, Sizedouble
    real, DIMENSION(:), ALLOCATABLE::aa !!! 11.6.2010
    integer, DIMENSION(:), ALLOCATABLE:: im
    REAL(8), DIMENSION(:), ALLOCATABLE::dd
!
!
    print help: <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    if (iargc().LT.3) then
    TEXTS
    write(*,*)
    write(*,*)'RandomForest modified by MJ during May and June 2010'
    write(*,*)' Fortran 90 version with dynamic memory allocation.'
    write(*,*)' Free under GNU General Public License.'
    write(*,*)' When published please cite as:'
    write(*,*)' -Breiman,L.: Random Forests, Machine Learning Vol. 45, No. 1, pp.
5-32.(2001)'
    write(*,*)' -Jirina,M., Jirina, M., jr.: Testing Random Forest for Unix and
```

Windows.
write(*,*)' Technical Report No. V-1075, Institue of Informatics AS CR, Prague,'

```
write(*,*)' Czech Republic (2010).'
```

    write (*,*)
    msg1='Usage: prog.exe datatrain datatest mdim '
    msg2 \(=\) ' [named parameters]'
    write(*,*)msg1,msg2
    msgl='DESCRIBE DATA (with optional defaults)'
    write(*,*)msgl
    msgl='Data files: numeric with class labels a'
    msg2='s integers, reals, or strings.'
    write(*,*) msg1,msg2
    msgl='datatrain: training data file name'
    write(*,*)msgl
    msg1='datatest: testing data file name'
    write(*,*) msg1
    msgl='mdim: task dimensionality (number of \(\mathrm{V}^{\prime}\)
    msg2='ariables)]'
    write(*,*) msg1,msg2
    msg1='Named parameters: Form: Parname=Parvalu'
    msg2='e.'
    write(*,*) msg1,msg2
    msgl='[ntrain: number of samples (cases) in t'
    msg2='he training data - NOT NECESSARY NOW!] '
    write(*,*) msg1,msg2
    msg1='[nclass: number of classes - NOT NECESS'
    msg2='ARY NOW!]'
    write (*, *) msg1, msg2
    msg1='[ntest: the number of samples (cases) i'
    msg2='n the test set. NOTE: Put ntest=1 if '
    write(*,*) msg1,msg2
    msgl=' there is no test set. Putting ntest=0'
    msg2=' may cause compiler complaints.'
    write (*, *) msg1, msg2
    msg1=' - NOT NECESSARY NOW if data test file'
    msg2=' exists!]'
    write(*,*)msg1,msg2
    msgl='maxcat: the largest number of values as'
    msg2='sumed by a categorical variable in data'
    write(*,*) msg1,msg2
    msg1='labeltr=1 if the data has class labels.'
    msg2=' If not, \(=1\) or 2 adds a synthetic class'
    write(*,*) msg1,msg2
    msg1='labelts=0 if the test set has no class '
    msg2='labels, 1 if the test set has cl.labels'
    write(*,*) msg1,msg2
    write (*,*)
    msgl='SET RUN PARAMETERS'
    write(*,*)msg1
        123567801234567890123456789012345678901
    msg1='mtry0=number of variables randomly sele'
    msg2='cted at each node'
    write (*, *) msg1, msg2
    msgl=' Default=int(sqrt(float(mdim))+0.5), (o'
    msg2='riginally 2 later on 5). Needs tuning!'
    write(*,*) msg1, msg2
    msgl=' Begin with this value and try a value '
    msg2='twice as high and half as low. See manu'
    write (*, *) msg1, msg2
    msgl='ndsize=1=minimum node size'
    write(*,*)msg1
    \(\mathrm{msg} 1=' j b t=500=\) number of trees to grow'
    write (*, *) msg1
    msg1='look=100=how often you want to check th'
    msg2='e prediction error'
    write (*, *) msg1,msg2
    msg1='lookcls=1=show on the screen or not'
    ```
write(*,*)msg1
msg1='jclasswt=0'
write(*,*)msg1
msg1='mdim2nd=0'
write(*,*)msg1
msg1='mselect=0'
write(*,*)msg1
write(*,*)
msg1='OUTPUT CONTROLS; all defaults zero:'
write(*,*)msg1
msg1='ROC=1 to get data for ROC curve i.e. ou'
msg2='tputs for each test sample else 0. (fil'
write(*,*)msg1,msg2,'e roc.txt; default 0.'
msg1='isumout= 0/1 1=summary to screen. In'
msg2='cludes err rates & confusion matrix'
write(*,*)msg1,msg2
msg1='idataout = 0/1/2 1=train,2=adds test'
msg2='(7)'
write(*,*)msg1,msg2
msg1='impfastout = 0/1 1=gini fastimp(8)'
write(*,*)msg1
msg1='impout = 0/1/2 1=imp,2=to screen(9)'
write(*,*)msg1
msgl='impnout = 0/1 1=impn (10)'
write(*,*)msg1
msg1='interout=0/1/2 1=interaction, 2=screen'
msg2='(11)'
write(*,*)msg1,msg2
msg1='iprotout=0/1/2 1=prototypes,2=screen'
msg2='(12)'
write(*,*)msg1,msg2
msg1='iproxout=0/1/2 1=prox,2=adds test (13)'
write(*,*)msg1
msg1='iscaleout = 0/1 1=scaling coors(14)'
write(*,*)msg1
msg1='ioutlierout=0/1/2 1=train,2=adds test'
msg2='(15)'
write(*,*)msg1,msg2
write(*,*)
msg1='NAME OUTPUT FILES FOR SAVING THE FOREST'
msg2=' STRUCTURE'
write(*,*)msg1,msg2
msg1='isaverf=1 savedforest'
write(*,*)msg1
msg1='isavepar=1 savedparams'
write(*,*)msg1
msgl='isavefill=1 savedmissfill'
write(*,*)msg1
msg1='isaveprox=1 savedprox'
write(*,*)msg1
write(*,*)
msg1='NAME OUTPUT FILES TO SAVE DATA FROM CUR'
msg2='RENT RUN'
write(*,*)msg1,msg2
msg1='idataout=1 save-data-from-run'
write(*,*)msg1
msg1='impfastout=1 save-impfast'
write(*,*)msg1
msgl='impout=1 save-importance-data'
write(*,*)msg1
msg1='impnout=1 save-caseimp-data'
write(*,*)msg1
msg1='interout=1 save-pairwise-effects'
write(*,*)msg1
msg1='iprotout=1 save-protos'
```

```
    write(*,*)msg1
    msg1='iproxout>=1 save-run-proximities'
    write(*,*)msg1
    msg1='iscaleout=1 save-scale'
    write(*,*)msg1
    msg1='ioutlierout>=1 save-outliers'
    write(*,*)msg1
    msg1='iviz=1 the graphics program is to be us'
    msg2='ed.'
    write(*,*)msg1,msg2
    write(*,*)
    stop
    endif
    DEFAULTS of fixed parameters: <<<<<<<<<<<<<<<<<<<<<
        datatrain='data.train'
        datatest='data.test'
        mdim=9
evaluate fixed parameters:
call getarg(1, datatrain)
call getarg(2, datatest)
call getarg(3, txt)
read(txt, '(I10)') mdim
```



```
    DEFAULTS: <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
        ntrain=169
        nclass=6
        maxcat=1
        ntest=-45
        labeltr=1
        labelts=1
ELIMINATE SOME OF "DEFAULTS" 28 May 2010
    write(*,*)'task dimension is ',mdim
ntrain=169 testing end of file: <<<<<<<<<<<<<<<<<<<<<<<
nclass=6 and setting the number of classes
open(34,file=datatrain,status='old')
    do n=1,100
        stri(n)=' '
    enddo
    stri2=' '
    nclass=0
    do n=1,10000
        read(34,*,END=11) (xx,m=1,mdim), stri1
        write(*,*)'stri1=',stril(1:1),'=xx'
            do j=1,25
            chr=stril(j:j)
            write(*,*)'chr=', chr,'=='
            read(chr,'(I1)',ERR=6)i !o to i tu vubec nejde, obvykle nula
        enddo
            if(j.EQ.1) then ! 9.6.2010
            j=2
        endif
        stri2=stri1(1:j-1)
        write(*,*)'j=',j,'x',stri2,'y'
        do i=1,100
            write(*,*)'i=',i,'x',stri2,'y',stri(i),'z'
            if(stri2.EQ.stri(i))then
                    write(*,*)j,'then','y',i
                    goto 10
            else
                    if(i.GT.nclass)then!new class is added
                    nclass=nclass+1
                    stri(nclass)=stri2
                    write(*,*) j,'else','y',nclass
```


## endif

## endif

 enddoenddo
close(34)
11 continue
ntrain=n-1
LLL=mylen (datatrain)
KKK=mylen(datatest)
write(*,*)'Learn:',datatrain(1:LLL),' test:',datatest(1:KKK)
msgl='No. of rows of the lrn file, no.of lab:'
write(*,*)msg1, ntrain, nclass
ntest=45 testing end of file: $\lll \lll \lll \lll \lll \lll \lll<$
open (34,file=datatest, status='old')
do $i=1,1000000$
$\operatorname{read}(34, *, E N D=20) \quad(\operatorname{stri}(j), j=1, m d i m)$
enddo
20 close (34)
ntest=i-1
msgl='No. of rows of the tst file is:'
write(*,*)msg1, ntest
$!$
OUTPUT CONTROLS
parameter(
isumout $=1!0 / 1 \quad 1=$ summary to screen
idataout $=0!0 / 1 / 2 \quad 1=$ train,2=adds test (7)
impfastout $=0!0 / 1 \quad 1=g i n i$ fastimp (8)
impout $=10!0 / 1 / 2^{1=i m p, 2=t o} \operatorname{screen}(9)$
impnout $=\quad 0!0 / 1 \quad 1=i m p n \quad$ (10)
interout $=\quad 0!0 / 1 / 2 \quad 1=$ interaction, $2=\operatorname{screen}(11)$
iprotout $=\quad 0!0 / 1 / 2 \quad 1=$ prototypes, $2=\operatorname{screen}(12)$
iproxout $=\quad 0!0 / 1 / 2 \quad 1=$ prox, $2=$ adds test (13)
iscaleout $=0!0 / 1 \quad 1=$ scaling coors (14)
ioutlierout $=0!0 / 1 / 2 \quad 1=$ train,2 $\quad$ adds test (15)
CONTROL PARAMETERS
parameter
SET RUN PARAMETERS
$\mathrm{ROC}=0$
mtry0 $=$ int (sqrt (float (mdim)) +0.5 ) ! 2 later on 5
write $(*, *)^{\prime} \operatorname{mtry} 0=$ ' , mtry 0
ndsize=1
$j b t=500$
look=100
lookcls=0
jclasswt=0
mdim2nd=0
mselect $=0$
jbt = number of trees to grow
mtry=number of variables randomly selected at each node
look=how often you want to check the prediction error
ndsize=minimum node sizec
SET IMPORTANCE OPTIONS
imp=1
interact $=0$
impn=1

```
SET PROXIMITY COMPUTATIONS
nprox=1
nrnn=ntrain
SET OPTIONS BASED ON PROXIMITIES
noutlier=0
nscale=0
nprot=0
REPLACE MISSING VALUES
code=-999
missfill=0
GRAPHICS
iviz=0
SAVING A FOREST
isaverf=0
isavepar=0
isavefill=0
isaveprox=0
RUNNING A SAVED FOREST
irunrf=0
ireadpar=0 !to read parameters, see line 1463 (starts with '888')
ireadfill=0
ireadprox=0
```



```
evaluate named parameters <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
    do i=4,iargc()
        call getarg(i, txt)
        j=len (txt)-1
        do while (txt(j:j) .ne. '=')
            j = j - 1
    enddo
    if (j.GT.0) then
            ParName=txt (1:j-1)
            k=1
            do while (txt(k:k) .ne. ' ')
                k = k + 1
            enddo
            parstr=txt(j+1:k-1)
            pb='ntrain'
            if(ParName.eq.pb) read(parstr, '(I10)') ntrain
            pb='nclass'
            if(ParName.eq.pb) read(parstr, '(I10)') nclass
            pb='maxcat'
            if(ParName.eq.pb) read(parstr, '(I10)') maxcat
            pb='ntest'
            if(ParName.eq.pb) read(parstr, '(I10)') ntest
            pb='labeltr'
            if(ParName.eq.pb) read(parstr, '(I10)') labeltr
            pb='labelts'
            if(ParName.eq.pb) read(parstr, '(I10)') labelts
            pb='mtry0'
            if(ParName.eq.pb) read(parstr, '(I10)') mtry0
            pb='lookcls'
            if(ParName.eq.pb) read(parstr, '(I10)') lookcls
            pb='isumout'
            if(ParName.eq.pb) read(parstr, '(I10)') isumout
            pb='idataout'
            if(ParName.eq.pb) read(parstr, '(I10)') idataout
            pb='impfastout'
            if(ParName.eq.pb) read(parstr, '(I10)') impfastout
            pb='impout'
```

```
    if(ParName.eq.pb) read(parstr, '(I10)') impout
    pb='impnout'
    if(ParName.eq.pb) read(parstr, '(I10)') impnout
    pb='interout'
    if(ParName.eq.pb) read(parstr, '(I10)') interout
    pb='iprotout'
    if(ParName.eq.pb) read(parstr, '(I10)') iprotout
    pb='iproxout'
    if(ParName.eq.pb) read(parstr, '(I10)') iproxout
    pb='iscaleout'
    if(ParName.eq.pb) read(parstr, '(I10)') iscaleout
    pb='ioutlierout'
    if(ParName.eq.pb) read(parstr, '(I10)') ioutlierout
    pb='isaverf'
    if(ParName.eq.pb) read(parstr, '(I10)') isaverf
    pb='isavepar'
    if(ParName.eq.pb) read(parstr, '(I10)') isavepar
    pb='isavefill'
    if(ParName.eq.pb) read(parstr, '(I10)') isavefill
    pb='isaveprox'
    if(ParName.eq.pb) read(parstr, '(I10)') isaveprox
    pb='idataout'
    if(ParName.eq.pb) read(parstr, '(I10)') idataout
    pb='impfastout'
    if(ParName.eq.pb) read(parstr, '(I10)') impfastout
    pb='impout'
    if(ParName.eq.pb) read(parstr, '(I10)') impout
    pb='impnout'
    if(ParName.eq.pb) read(parstr, '(I10)') impnout
    pb='interout'
    if(ParName.eq.pb) read(parstr, '(I10)') interout
    pb='iprotout'
    if(ParName.eq.pb) read(parstr, '(I10)') iprotout
    pb='iproxout'
    if(ParName.eq.pb) read(parstr, '(I10)') iproxout
    pb='iscaleout'
    if(ParName.eq.pb) read(parstr, '(I10)') iscaleout
    pb='ioutlierout'
    if(ParName.eq.pb) read(parstr, '(I10)') ioutlierout
    pb='iviz'
    if(ParName.eq.pb) read(parstr, '(I10)') iviz
    pb='jbt'
    if(ParName.eq.pb) read(parstr, '(I10)') jbt
    pb='ROC'
    if(ParName.eq.pb) read(parstr, '(I10)') ROC
        endif
    enddo
    msg1=datatrain
    msg2=datatest
    write(*,*)msg1,msg2
    write(*,*) nclass
    write(*,*)
DERIVED PARAMETERS (DO NOT CHANGE)
parameter(
    nsample=(2-labeltr)*ntrain
    nrnodes=2*nsample+1
    mimp=imp* (mdim-1)+1
    ifprot=nprot/(nprot-.1)
```

```
    ifscale=nscale/(nscale-.1)
    iftest=ntest/(ntest-.1)
    nprot0=(1-ifprot) +nprot
    nscale0=(1-ifscale)+nscale
    ntest0=(1-iftest)+ntest
    mdim0=interact *(mdim-1)+1
    near=nprox* (nsample-1) +1
============================================================
********* ARRAYS FOR THE MAIN2 SUBROUTINE ************
============================================================
compute beginnings of REAL arrays
    II x=1
    IIxts=IIx+mdim*nsample
    IIv5=IIxts+mdim*ntest0
    IIv95=IIv5+mdim
    IItgini=IIv95+mdim
    IIzt=IItgini+mdim
    IIavgini=IIzt+mdim
    IIvotes=IIavgini+mdim
    IIeffect=IIvotes+mdim0* jbt
    IIteffect=IIeffect+mdim0*mdim0
    IIhist=IIteffect+mdim0*mdim0
    IIg=IIhist+(1+mdim0)*mdim0
    IIfill=IIg+mdim0
    IIrinpop=IIfill+mdim
    IIdgini=IIrinpop+near* jbt
    IIxbestsplit=IIdgini+nrnodes
    IItnodewt=IIxbestsplit+nrnodes
    IItw=IItnodewt+nrnodes
    IItn=IItw+nrnodes
    IIv=IItn+nrnodes
    IIwin=IIv+nsample
    IItemp=IIwin+nsample
    IIq=IItemp+nrnn
    IIdevout=IIq+nclass*nsample
    IIclasswt=IIdevout+nclass
    IIwr=IIclasswt+nclass
    IItmissts=IIwr+nclass
    IItmiss=IItmissts+nclass
    IItclasspop=IItmiss+nclass
    IIwl=IItclasspop+nclass
    IIrmedout=IIwl+nclass
    IItclasscat=IIrmedout+nclass
    IIqts=IItclasscat+nclass*maxcat
    IIclasspop=IIqts+nclass*ntest0
    IIsignif=IIclasspop+nclass*nrnodes
    IIzscore=IIsignif+mimp
    IIsqsd=IIzscore+mimp
    IIavimp=IIsqsd+mimp
    IIqimp=II avimp+mimp
    IIqimpm=IIqimp+nsample
    IItout=IIqimpm+nsample*mimp
    IIouttr=IItout+near
    IIxc=IIouttr+near
    IIdn=IIxc+maxcat
    IIcp=IIdn+maxcat
    IIcm=IIcp+maxcat
    IIvotecat=IIcm+maxcat
    IIfreq=IIvotecat+maxcat
    IIwc=IIfreq+maxcat
    IIoutts=IIwc+nsample
    IIpopclass=IIoutts+ntest0
    IIprotlow=IIpopclass+nprot0*nclass
    IIprot=IIprotlow+mdim*nprot0*nclass
    IIprothigh=IIprot+mdim*nprot0*nclass
    IIprotfreq=IIprothigh+mdim*nprot0*nclass
```

```
    IIrpop=IIprotfreq+mdim*nprot0*nclass*maxcat
    IIprotv=IIrpop+nrnodes
    IIwtx=IIprotv+mdim*nprot0*nclass
    IIprotvlow=IIwtx+nsample
    IIprotvhigh=IIprotvlow+mdim*nprot0*nclass
    Sizereal=IIprotvhigh+mdim*nprot0*nclass
compute beginnings of INTEGER arrays
    IIcat=1
    IIiv=IIcat+mdim
    IImsm=IIiv+mdim
    IImuse=IImsm+mdim
    IIirnk=IImuse+mdim
    IImissing=IIirnk+mdim*jbt
    IIa=IImissing+mdim*near
    IIasave=IIa+mdim*nsample
    IIb=IIasave+mdim*nsample
    IIcl=IIb+mdim*nsample
    IIout=IIcl+nsample
    IInodextr=IIout+nsample
    IInodexvr=IInodextr+nsample
    IIjin=IInodexvr+nsample
    IIjoob=IIjin+nsample
    IIpjoob=IIjoob+nsample
    IIndbegin=IIpjoob+nsample
    IIjvr=IIndbegin+near*jbt
    IIjtr=IIjvr+nsample
    IIjest=IIjtr+nsample
    IIibest=IIjest+nsample
    IIisort=IIibest+nrnn
    IIloz=IIisort+nsample
    IIta=IIloz+near*nrnn
    IIncase=IIta+nsample
    IIidmove=IIncase+nsample
    IIkpop=IIidmove+nsample
    IIjests=IIkpop+nrnodes
    IIjts=IIjests+ntest0
    IIiwork=IIjts+ntest0
    IInodexts=IIiwork+near
    IIclts=IInodexts+ntest0
    IIimax=IIclts+ntest0
    IIjinb=IIimax+ntest0
    IIbestsplitnext=IIjinb+near*jbt
    IIbestvar=IIbestsplitnext+nrnodes
    IIbestsplit=IIbestvar+nrnodes
    IInodestatus=IIbestsplit+nrnodes
    IInodepop=IInodestatus+nrnodes
    IInodestart=IInodepop+nrnodes
    IInodeclass=IInodestart+nrnodes
    IIparent=IInodeclass+nrnodes
    IItreemap=IIparent+nrnodes
    IIncts=IItreemap+2*nrnodes
    IInc=IIncts+nclass
    IImtab=IInc+nclass
    IIncn=IImtab+nclass*nclass
    IIits=IIncn+near
    IIjpur=IIits+nsample
    IInpend=II jpur+nrnn
    IIinear=IInpend+nclass
    IInrcat=IIinear+nrnn
    IIkcat=IInrcat+maxcat
    IIncatsplit=IIkcat+maxcat
    IInbestcat=IIncatsplit+maxcat
    IIncp=IInbestcat+maxcat*nrnodes
    IInodexb=IIncp+near
    IInpcase=IInodexb+near* jbt
    IIncount=IInpcase+near*jbt
    IInod=IIncount+near*jbt
```

```
            Sizeint=IInod+nrnodes
!
! compute beginnings of double precision arrays
            KKprox=1
            KKy=KKprox+near*nrnn
            KKu=KKy+near
            KKdl=KKu+near
            KKxsc=KKdl+nscale0
            KKred=KKxsc+near*nscale0
            KKee=KKred+near
            KKev=KKee+near
            KKppr=KKev+near*nscale0
            Sizedouble=KKppr+near
            test the total length
!
            write(*,*)'Sizereal=',Sizereal,' Sizeint=',Sizeint,&
                Sizedouble=',Sizedouble
!!! 11.6.2010
ALLOCATE ( aa(Sizereal) ) ! Allocate heap space.
ALLOCATE ( im(Sizeint) ) ! Allocate heap space.
ALLOCATE ( dd(Sizedouble) ) ! Allocate heap space.
!
!
!
!
!
    ========================================================
    *************** CALLING MAIN2 SUBROUTINE *****************
    ===========================================================
    call main2(datatrain, datatest,&
    mdim, ntrain,nclass,maxcat,ntest, &
    labelts,labeltr,mtry0, ndsize,jbt,look,lookcls,&
    jclasswt,mdim2nd,mselect,imp,interact,impn,&
    nprox,nrnn, noutlier,nscale, nprot,missfill,iviz, &
    isaverf,isavepar,isavefill,isaveprox,&
    irunrf,ireadpar,ireadfill,ireadprox, &
    isumout,idataout,impfastout,impout,impnout,interout,&
    iprotout,iproxout,iscaleout,ioutlierout,&
    nsample, nrnodes,mimp,near, &
    ifprot,ifscale,iftest,mdim0, ntest0, nprot0, nscale0,&
! My control:
            ROC,&
        REAL arrays
    aa(IIx),&
    aa(IIxts),&
    aa(IIv5),&
    aa(IIv95),&
    aa(IItgini),&
    aa(IIzt),&
    aa(IIavgini),&
    aa(IIvotes),&
    aa(IIeffect),&
    aa(IIteffect),&
    aa(IIhist),&
    aa(IIg),&
    aa(IIfill),&
    aa(IIrinpop),&
    aa(IIdgini),&
    aa(IIxbestsplit),&
    aa(IItnodewt),&
    aa(IItw),&
    aa(IItn),&
    aa(IIv),&
    aa(IIwin),&
    aa(IItemp),&
    aa(IIq),&
    aa(IIdevout),&
    aa(IIclasswt),&
    aa(IIwr),&
    aa(IItmissts),&
```

```
aa(IItmiss),&
aa(IItclasspop),&
aa(IIwl),&
aa(IIrmedout),&
aa(IItclasscat),&
aa(IIqts),&
aa(IIclasspop),&
aa(IIsignif),&
aa(IIzscore),&
aa(IIsqsd),&
aa(IIavimp), &
aa(IIqimp),&
aa(IIqimpm),&
aa(IItout),&
aa(IIouttr),&
aa(IIxc),&
aa(IIdn), &
aa(IIcp),&
aa(IIcm),&
aa(IIvotecat),&
aa(IIfreq),&
aa(IIwc),&
aa(IIoutts),&
aa(IIpopclass),&
aa(IIprotlow),&
aa(IIprot),&
aa(IIprothigh),&
aa(IIprotfreq),&
aa(IIrpop),&
aa(IIprotv),&
aa(IIwtx),&
aa(IIprotvlow),&
aa(IIprotvhigh),&
    INTEGER arrays
im(IIcat),&
im(IIiv),&
im(IImsm),&
im(IImuse),&
im(IIirnk),&
im(IImissing),&
im(IIa),&
im(IIasave),&
im(IIb),&
im(IIcl),&
im(IIout),&
im(IInodextr), &
im(IInodexvr),&
im(IIjin),&
im(IIjoob),&
im(IIpjoob),&
im(IIndbegin), &
im(IIjvr),&
im(IIjtr),&
im(IIjest),&
im(IIibest),&
im(IIisort),&
im(IIloz),&
im(IIta),&
im(IIncase),&
im(IIidmove),&
im(IIkpop),&
im(IIjests),&
im(IIjts),&
im(IIiwork),&
im(IInodexts),&
im(IIclts),&
im(IIimax),&
```

```
    im(IIjinb),&
    im(IIbestsplitnext),&
    im(IIbestvar),&
    im(IIbestsplit),&
    im(IInodestatus),&
    im(IInodepop),&
    im(IInodestart),&
    im(IInodeclass),&
    im(IIparent),&
    im(IItreemap),&
    im(IIncts),&
    im(IInc),&
    im(IImtab),&
    im(IIncn),&
    im(IIits),&
    im(IIjpur),&
    im(IInpend),&
    im(IIinear),&
    im(IInrcat),&
    im(IIkcat),&
    im(IIncatsplit),&
    im(IInbestcat),&
    im(IIncp),&
    im(IInodexb),&
    im(IInpcase),&
    im(IIncount),&
    im(IInod),&
!
!!! DOUBLE precision arrays
    dd(KKprox),&
    dd(KKy),&
    dd(KKu),&
    dd(KKdl),&
    dd(KKxsc),&
    dd(KKred),&
    dd(KKee), &
    dd(KKev),&
    dd(KKppr) &
    )
!
    return
    end ! OF THE MAIN
!
!
!
!
!
!
! ==============================================================
! ******* MAIN REWRITTEN AS A SUBROUTINE MAIN2 *********
! =============================================================
!
    subroutine main2(datatrain, datatest, &
    mdim,ntrain,nclass,maxcat, ntest, &
    labelts,labeltr,mtry0,ndsize,jbt,look,lookcls,&
    jclasswt,mdim2nd,mselect,imp,interact,impn, &
    nprox,nrnn,noutlier,nscale, nprot,missfill,iviz, &
    isaverf,isavepar,isavefill,isaveprox,&
    irunrf,ireadpar,ireadfill,ireadprox,&
    isumout,idataout,impfastout,impout,impnout,interout,&
    iprotout,iproxout,iscaleout,ioutlierout, &
    nsample,nrnodes,mimp, near,&
    ifprot,ifscale,iftest,mdim0,ntest0, nprot0,nscale0,&
! my control:
    ROC,&
! real arrays
    x,xts,v5,v95,tgini,zt,avgini, &
    votes,effect,teffect,hist,g,fill,rinpop, &
```

```
    dgini,xbestsplit,tnodewt,&
    tw,tn,v,win,temp,q, devout,classwt,wr,&
    tmissts,tmiss,tclasspop,wl,rmedout,tclasscat,qts,&
    classpop,signif,zscore,sqsd,avimp,qimp,qimpm,tout,&
    outtr, xc,dn,cp,cm,votecat,freq,wc,outts,&
    popclass,protlow, prot,prothigh,protfreq,rpop,&
    protv,wtx,protvlow, protvhigh,&
! integer arrays
    cat,iv,msm,&
    muse,irnk,missing,a,&
    asave,b,&
    cl,out,nodextr,nodexvr,&
    jin,joob, pjoob,ndbegin,&
    jvr,jtr,jest,ibest,&
    isort,loz,&
    ta,ncase,idmove,kpop,&
    jests,jts,iwork,&
    nodexts,clts,imax,jinb,&
    bestsplitnext,bestvar,bestsplit,&
    nodestatus, nodepop, nodestart,&
    nodeclass,parent,treemap,&
    ncts,nc,mtab,ncn, &
    its, jpur, npend,inear, &
    nrcat,kcat,ncatsplit,&
    nbestcat, ncp, nodexb,&
    npcase,ncount,nod,&
! DOUBLE precision arrays
    prox,&
    y,&
    u,&
    dl,&
    xsc,&
    red,&
    ee,&
    ev,&
    ppr&
    )
!
    character*255 datatrain, datatest
!
    integer mdim,ntrain,nclass,maxcat,ntest,&
    labelts,labeltr,mtry0,ndsize,jbt,look,lookcls,&
    jclasswt,mdim2nd,mselect,imp,interact,impn,&
    nprox,nrnn,noutlier,nscale,nprot,missfill,iviz,&
    isaverf,isavepar,isavefill,isaveprox,&
    irunrf,ireadpar,ireadfill,ireadprox,&
    isumout,idataout,impfastout,impout,impnout,interout,&
    iprotout,iproxout,iscaleout,ioutlierout,&
    nsample, nrnodes,mimp,near,&
    ifprot,ifscale,iftest,mdim0,ntest0,nprot0,nscale0,&
        ROC
    character*25 stri(100),stri1,stri2 !27.5.2010
    character chr
    real code,xx
    DIMENSIONING OF ARRAYS
    real x(mdim,nsample), xts(mdim,ntest0),v5(mdim),v95(mdim),&
    tgini(mdim), zt(mdim), avgini(mdim),&
    votes(mdim0,jbt), effect(mdim0,mdim0),teffect (mdim0,mdim0),&
    hist(0:mdim0,mdim0),g(mdim0),fill(mdim),rinpop(near, jbt), &
    dgini(nrnodes), xbestsplit(nrnodes),tnodewt(nrnodes),&
    tw(nrnodes),tn(nrnodes),v(nsample),win(nsample),temp(nrnn),&
    q(nclass,nsample), devout(nclass), classwt (nclass),wr(nclass),&
    tmissts(nclass),tmiss(nclass),tclasspop(nclass),wl(nclass),&
    rmedout(nclass),tclasscat(nclass,maxcat),qts(nclass,ntest0),&
```

```
classpop(nclass,nrnodes),signif(mimp), zscore(mimp),sqsd(mimp),&
avimp(mimp), qimp(nsample),qimpm(nsample,mimp), tout(near), &
outtr(near), xc (maxcat),dn(maxcat), cp (maxcat), cm(maxcat) ,&
votecat(maxcat), freq(maxcat),wc(nsample), outts(ntest0),&
popclass(nprot0,nclass),protlow(mdim, nprot0,nclass), &
prot(mdim,nprot0,nclass), prothigh(mdim, nprot0, nclass),&
protfreq(mdim, nprot0,nclass,maxcat), rpop(nrnodes),&
protv(mdim, nprot0,nclass),wtx(nsample),&
protvlow(mdim, nprot0,nclass), protvhigh(mdim, nprot0, nclass)
integer cat(mdim),iv(mdim),msm(mdim),&
    muse(mdim),irnk(mdim,jbt),missing(mdim,near),a(mdim,nsample),&
asave(mdim,nsample),b(mdim,nsample), &
cl(nsample), out(nsample), nodextr(nsample), nodexvr(nsample),&
jin(nsample),joob(nsample), pjoob(nsample), ndbegin(near, jbt), &
jvr(nsample),jtr(nsample),jest(nsample),ibest(nrnn), &
isort(nsample),loz(near,nrnn), &
ta(nsample),ncase(nsample),idmove(nsample),kpop(nrnodes),&
jests(ntest0),jts(ntest0),iwork(near),&
nodexts(ntest0),clts(ntest0),imax(ntest0),jinb(near,jbt),&
bestsplitnext(nrnodes), bestvar(nrnodes),bestsplit(nrnodes), &
nodestatus(nrnodes), nodepop(nrnodes), nodestart(nrnodes), &
nodeclass(nrnodes), parent(nrnodes),treemap (2, nrnodes), &
ncts(nclass),nc(nclass),mtab(nclass,nclass),ncn(near),&
    its(nsample), jpur(nrnn), npend(nclass),inear(nrnn),&
nrcat(maxcat),kcat(maxcat), ncatsplit(maxcat), &
nbestcat (maxcat, nrnodes), ncp (near), nodexb (near, jbt), &
npcase(near, jbt), ncount (near, jbt), nod(nrnodes), nmfmax, &
ncsplit,ncmax,nmissfill, ndimreps,nmf,nmd,iseed
USED IN PROXIMITY AND SCALING
double precision prox(near,nrnn),y(near),u(near), &
dl(nscale0), xsc(near,nscale0),red(near), ee(near),&
ev(near,nscale0),ppr(near)
character*500 text
SCALAR DECLARATIONS
real errtr,errts,tavg,er,randomu
integer mtry,n,m,mdimt,k,j,i,m1,jb,nuse,ndbigtree,jj,mr,&
n0,n1,n2,n3,n4,n5,n6,n7
write(*,*)'after arrays declaration'
READ OLD TREE STRUCTURE AND/OR PARAMETERS
[The original continues. The modified procedure for reading data follows:]
write(*,*)' READ IN DATA--SEE MANUAL FOR FORMAT 27 May 2010'
do n=1,100
    stri(n)=' '!!! <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
enddo
stri2=' '
nclass=0
open(16, file=datatrain, status='old')
do n=1,ntrain
    read(16,*) (x(m,n),m=1,mdim), stri1
    write(*,*)'stril=',stril(1:1),'=xx'
        do j=1,25
            chr=stril(j:j)
            read(chr,'(I1)',ERR=56)i !o to i tu vubec nejde, obvykle nula
```

```
            enddo
    5 6
        if(j.EQ.1) then ! 9.6.2010
        j=2
            endif
            stri2=stri1(1:j-1)
            write(*,*)j,'x',stri2,'y'
            do i=1,100
                if(stri2.EQ.stri(i))then
                cl(n)=i
                write(*,*) j,'then',stri2,'y',cl(n)
                        goto 60
                else
                                    if(i.GT.nclass)then!new class is added
                                    nclass=nclass+1
                                    stri(nclass)=stri2
                                    cl(n)=nclass
                                    write(*,*)j,'else',stri2,'y',cl(n)
                                    goto 60
                endif
                endif
            enddo
            write(*,*)cl(n)
                continue
            enddo
            close(16)
            write(*,*)'final classes=',nclass
        if(ntest.gt.0) then
            open(17, file=datatest, status='old')
            if(labelts.ne.0) then
                do n=1, ntest0
                    read(17,*) (xts(m,n),m=1,mdim), stri1 !xx !<<<<<<<<<<<<TADY
                    write(*,*)'stril=',stril(1:1),'=xx'
                                    do j=1,25
                                    chr=stri1(j:j)
                                    read(chr,'(I1)',ERR=66)i !o to i tu vubec nejde,
obvykle nula
    6 6
!
!
!
    70
!
```

! SELECT SUBSET OF VARIABLES TO USE
[The original continues till the end. After it we added a simple function as follows.]

```
    function mylen(str) ! 11 June 2010
            Computes length of a string not counting spaces from the right.
            Two spaces between other charactes are admissible.
            character*1 str
            mezcount=0
            lastpos=0
            do i=1,500
            if(str(i:i).eq.' ') then
                    mezcount=mezcount+1
                    if(mezcount.GE.3) then
                        mylen=lastpos
                    return
                    endif
            else
                lastpos=i
                mezcount=0
            endif
        enddo
    end
```

!


[^0]:    ${ }^{1}$ [ ] Breiman, L.: Random Forests, Machine Learning Vol. 45, No. 1, pp. 5-32. (2001)
    [ ] Jiřina, M., Jiřina, M., jr.: Testing Random Forest for Unix and Windows. Technical Report No. V-1075 , Institue of Informatics AS CR, Prague, Czech Republic. (2010)

[^1]:    ${ }^{2}$ In our command line version just this is used as the default value.

