Adapting GNU Random Forest program for Unix and Windows[1]

DRAFT

Marcel Jirina[a], M. Said Krayem[b], and Marcel Jirina, Jr.[c]

[a]*Institute of Computer Science AS CR, Pod Vodarenskou vezi 2, 182 07 Prague, Czech Republic marcel@cs.cas.cz*
[b]*Department of computer engineering, Faculty of electrical and elektronics engineering, University of Aleppo, Aleppo - P.O.Box 7072, Syria, drsaid@seznam.cz*
[c]*Faculty of Biomedical Engineering, Czech Technical University in Prague, Nám. Sítná 3105, 272 01, Kladno, Czech Republic, jirina@fbmi.cvut.cz*

Abstract. The Random Forest is a well-known method and also a program for data clustering and classification. Unfortunately, the original Random Forest program is rather difficult to use. Here we describe a new version of this program originally written in Fortran 77. The modified program in Fortran 95 needs to be compiled only once and information for different tasks is passed with help of arguments. The program was tested with 24 data sets from UCI MLR and results are available on the net.

Keywords: Program recycling; software maintenance; Random Forest; multivariate data; clustering; classification.
PACS: 62H30; 68T10; 68N19; 97P50

INTRODUCTION

The Random Forest [1], [2] is a method and also a program for data mining and pattern recognition. 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.
The message of this report is a new version of the famous RandForest program written by Leo Breiman and Adele Cutler [3] in Fortran 77. In the original version, all the information about data to be processed and details of processing must be included in the program source text. Thus, a modified program must be compiled and afterwards run with data.
In our modification, users need not compile the program. Information for different tasks is passed to the program with the help of arguments. The program has one source text and binaries can run under UNIX/LINUX or under Windows environment. A brief description of the problems of software maintenance, updating, and recycling is given first, then the "technology" of the modification of RandForest program is shown. The next Chapter describes testing data corpora used and gives results of processing by the Random Forest program.
The program and results are free under GNU General Public License [6].

RECYCLING SOFTWARE

A lot of old but quality software exists written under different environments and also with purposes different from those that exist and are needed today.
Once software is put into use, new requirements emerge and existing requirements change. Business changes often generate new requirements for existing software. Part of the software may have to be modified to correct errors that are found in operation, to adapt it for a new platform and to improve its performance or other non-functional

[1] The final version published: 11th International Conference of Numerical Analysis and Applied Mathematics 2013
AIP Conf. Proc. 1558, pp. 337-340 (2013); doi: 10.1063/1.4825492
AIP Publishing LLC 978-0-7354-1184-5 (2013). Available at: http://scitation.aip.org/content/aip/proceeding/aipcp/1558.

characteristics. Software development, therefore, does not stop when a program is finished or a system is delivered but continues throughout the lifetime of the system.
Software evolution is important because organizations are now completely dependant on their software systems and have invested millions in such systems. Their systems are critical business assets and money has to be invested in system change to maintain the value of these assets. A great part software budgets in large companies is, therefore, earmarked for maintaining existing systems, and we should not be surprised by figures, such as those given by Erlikh, suggesting that 90% of software costs are evolution costs. Consequently, one can think of software engineering as a spiral process with requirements, design, implementation and testing going on throughout the lifetime of the system.
When the transition from development to evolution is not seamless, the process of changing the software after delivery is often called software maintenance. Maintenance involves extra process activities, such as program comprehension, in addition to the routine activities of software development.
The Random Forest is a typical example:

- It is a very efficient program in its category (data classifiers)
- It was written in Fortran 77
- It was probably meant for manual experimentation only; even though it appeared on the net and was found useful in some research [11].

The first item represents an advantage, the second is no fault, really, but the third makes program rather difficult to use. As we describe later, it is necessary to set up (change) parameters in the source text of the Fortran program with all consequences in compiling and tuning.

RANDOM FOREST PROGRAM

Random Forest Approach

A basic idea of Random Forests is in short and best described in [1]. We cite here verbatim: "Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost (Freund and Schapire [1996]), but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance."

Motivation

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. This, however, may prove to be a stressful task for users who have no knowledge of Fortran 77 programming language and the peculiarities of the use of a compiler. This, in turn, may result in the use of another classifier that is friendlier to use.
The original program has 3435 lines and, in fact, consists of sophisticated and complex mathematical procedures. In such procedures no changes are advisable. So, changes of the program concentrate on the "environment" of these procedures. Now the program has 4337 lines and all additions and changes represent approximately 1100 new lines.
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 its parameters form a command line. 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. Named arguments constitute another possibility. Named argument consists of a name and a value of a parameter. Some separator, usually a colon or equivalence sign, can be found between the name and the value.

Transformation Concept

To change rather tough structure of the original RandFor program into more flexible needs consideration of

1. A dynamic memory allocation
2. Control with the use of command line attributes
3. Changing numerical class marks to strings
4. Add reporting files for recording of results of the batch runs and of detailed results.

To assure compatibility with UNIX/Linux and Windows environment it suffices to avoid all software or hardware add-ons. Fortunately, the original RandFor program satisfies this condition.

Ad 1. In Fortran 77, a feature of dynamic allocation does not exist. On the other hand, there is a feature 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 program into subroutine without changing any statement and any declaration of arrays and variables. The cost of 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 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. Belonging to the same source program are the run-time binaries for running under Windows or UNIX/Linux environment.

Ad 2. The most important step was the change of all Fortran "parameters" (statement parameter (…)) into variables. For this, the declaration of parameters as variables has to be separated from setting up their default values. To enable users to change parameter's value the analysis of the command line attributes has been included.

Ad 3. To allow class marks to have a form of a text, the analysis of class marks in the learning file and a transformation of them into numbers 1, 2 … was included at the beginning of the program. In a corresponding way, the procedures for reading training and testing data files (originally with fixed names data.train and data.test) have been modified accordingly.

Ad 4. We added two reporting files for recording of results of the batch runs. One of them reports about classification error for a pair of learning and testing files. The other optional file reports classification result of each sample of the testing set.

Changes of the RandFor program

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 the arrays and of all the 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. To construct true parameters for arrays, three vast integer, a real, and double precision arrays have been declared. The individual arrays in MAIN2 subroutine correspond to some part of one of these vast arrays of the same type (integer, real, double precision). These individual arrays form true parameters dynamically allocated at the time when subroutine MAIN2 is called.

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

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

Register.txt file. At the end of the 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. It is never rewritten. The line consists of names of training and testing files, the dimensionality of the task, and the number of classes, the learning classification error, and the testing classification error.

ROC.txt file If parameter ROC is set up to 1 (i.e. attribute ROC=1 appears in the command line), then a matrix of class probabilities together with the true class number and the class number found by the program appear as a file "ROC.txt". In the first line, the names of the training and testing files appear in greater detail, the second line gives headings for data on next lines. The third and other lines of this file give the true class number, the class number found by the program, 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]. It must be renamed before new run of RandFor program rewrites it.

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 by other methods. Our task is to present reproducible results and any comparisons are up to the kind reader, see [4], [12].

CONCLUSION

We have found that the original Random Forest program is 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. However, this may be a stressful task for users who have no knowledge of the FORTRAN programming language and of the peculiarities of using a compiler. This may result in the application of another classifier that is friendlier to use.

There is a feature of the FORTRAN language 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 a large part of the main program into subroutine without changing any declaration of arrays and variables. The cost of it is that the names of all the arrays and of all the variables shared with the main program must appear as formal parameters of such 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. Belonging to the same source program are the run-time versions of binaries for Windows or UNIX/Linux environment [12].

ACKNOWLEDGEMENTS

This work was supported by Technology Agency CR under project of series ALFA No. TA01010490. The work was also supported by Faculty of Biomedical Engineering of the Czech Technical University in Prague, RVO: 68407700, Czech Republic.

REFERENCES

1. L. Breiman: Random Forests, Machine Learning Vol. 45, No. 1, pp. 5-32. (2001)
2. L. Breiman: Manual On Setting Up, Using, And Understanding Random Forests V3.1. Statistics Department University of California Berkeley, CA, USA, 2002. Available also on-line http://oz.berkeley.edu/users/breiman/Using_random _forests_V3.1.pdf (read March 2013)
3. L. Breiman, and A. Cutler: The original Fortran code of the RandFor. Statistics Department University of California Berkeley, CA, USA, 2002. Available also on-line http://oz.berkeley.edu/users/breiman /RandomForests/cc_home.htm (read 26.5.2010).
4. M. Jiřina, M. Jiřina, jr.: Technical Report No. V-1075 , Inst. of Computer Science AS CR, Prague, Czech Republic. (2010).
5. R. Paredes: CPW: Class and Prototype Weights learning, [online], 2008, Available: <http://www.dsic.upv.es/~rparedes /research/CPW/index.html>.(read November, 2012)
6. GNU General Public License. On-line http://www.gnu.org/licenses /gpl.html (read June 2012)
7. T. Fawcett: ROC Graphs: Notes and Practical Considerations for Researchers. Machine Learning, (2004).
8. A. Frank, A. Asuncion, (2010). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. (Read March 2013)
9. J.H. Friedmann: Flexible Metric Nearest Neighbor Classification. Technical Report 113, Dept. of Statistics, Stanford University, 1994.
10. R. Paredes: 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 June 2012]
11. R.K. Bock et al. Nuclear Instruments and Methods in Physics Research. A, 2004, Vol. 516, -, pp. 511-528. ISSN 0168-9002.
12. M. Jiřina, M. Jiřina, jr.: Modified Random Forests Software. On-line http://www.marceljirina.cz/files/modified-random-forests.zip (2010).