Random Jungle, version 2.0.0

A powerful Random Forests (TM) implementation
Edition 2.0.0, 12 March 2013

by Daniel F. Schwarz et al. (Developer) to version 1.2.365
by Jochen Kruppa (Maintainer/Developer) to version 2.0.0
(randomjungle@googlegroups.com)

This manual (12 March 2013) is for Random Jungle (RJ) version 2.0.0, a package containing an implementation of the Random Forests (TM) method.

Table of Contents

1 Introduction and preliminaries

This first chapter explains what `Random Jungle` is, where `Random Jungle` comes from, how to read and use this documentation, how to call the `Random Jungle` program, and how to report bugs about it. It concludes by giving tips for reading the remainder of the manual.

Okay, I have no time to read, how do I get started? Go to the Chapter 5 [Application], page 15, where you get all the needed information on general Random Forest (TM) tasks. If you not found what you need, read the options list Section 2.1 [Operation modes], page 3 and finally write an e-mail to `randomjungle@googlegroups.com`.

The following chapters then detail all the features of the `Random Jungle` .

1.1 Introduction to `Random Jungle`

`Random Jungle` is an implementation of Random Forests (TM). It is supposed to analyse high dimensional data. In genetics, it can be used for analysing big Genome Wide Association (GWA) data. Random Forests (TM) is a powerful machine learning method. Most interesting features are variable selection, missing value imputation, classifier creation, generalization error estimation and sample proximities between pairs of cases.

`Random Jungle` is mostly compatible with the Linux and Windows (TM). Also, there exists compatibilities with Solaris.

1.2 Historical references

First ideas where seeded in 2006 during GAW15 workshop in Florida. In the beginning of 2008, Daniel F. Schwarz released 0.5.0 and 0.5.1 which was a "way pre-release" and fast, but lagged of features and documentation. However, the williams award was won at the International Genetic Epidemiology Society (IGES) converence 2008 in St. Louis with release 0.5.2. In late 2008 and 2009, intensive work improved the platform compatibility, added documentation and raised the number of features of `Random Jungle`.

1.3 Problems and bugs

If you have problems with `Random Jungle` or think you've found a bug, please report it. Before reporting a bug, make sure you've actually found a real bug. Carefully reread the documentation and see if it really says you can do what you're trying to do. If it's not clear whether you should be able to do something or not, report that too; it's a bug in the documentation!

Before reporting a bug or trying to fix it yourself, try to isolate it to the smallest possible input file that reproduces the problem. Then send us the input file and the exact results `Random Jungle` gave you. Also say what you expected to occur; this will help us decide whether the problem was really in the documentation.

Once you've got a precise problem, send e-mail to `randomjungle@googlegroups.com`. Please include the version number of `Random Jungle` you are using. You can get this information with the command *`rjungle --version`* or *`rjungle -Z`*.

Non-bug suggestions are always welcome as well. If you have questions about things that are unclear in the documentation or are just obscure features, please report them too.

1.4 Using this manual

This manual contains a number of examples of `Random Jungle` input and output, and a simple notation is used to distinguish input, output and error messages from `Random Jungle`. Examples are set out from the normal text, and shown in a fixed width font, like this

```
This is an example of an example!
```

To distinguish input from output, all output from `Random Jungle` is prefixed by the string '⇒', and all error messages by the string ' error '. When showing how command line options affect matters, the command line is shown with a prompt '`$ like this`', otherwise, you can assume that a simple *`rjungle`* invocation will work. Thus:

```
$ command line to invoke rjungle
Example of input line
⇒Output line from rjungle
error and an error message
```

The sequence '`^D`' in an example indicates the end of the input file. The sequence '`NL`' refers to the newline character. The majority of these examples are self-contained, and you can run them with similar results by invoking *`rjungle -d`*. In fact, the testsuite that is bundled in the Random Jungle package consists of the examples in this document! Some of the examples assume that your current directory is located where you unpacked the installation, so if you plan on following along, you may find it helpful to do this now:

```
$ cd randomjungle-2.0.0
```

2 Invoking Random Jungle

The format of the Random Jungle command is:

```
rjungle [option...]
```

or

```
rjunglesparse [option...]
```

All options begin with '-', or if long option names are used, with '--'. On some platforms long options might not work. A long option name need not be written completely, any unambiguous prefix is sufficient. POSIX requires Random Jungle to recognize arguments intermixed with files, even when POSIXLY_CORRECT is set in the environment. Most options take effect at startup regardless of their position, but some are documented below as taking effect after any files that occurred earlier in the command line. The argument '--' is a marker to denote the end of options.

With short options, options that do not take arguments may be combined into a single command line argument with subsequent options, options with mandatory arguments may be provided either as a single command line argument or as two arguments, and options with optional arguments must be provided as a single argument. In other words, *rjungle -QPDfoo -d a -df* is equivalent to *rjungle -Q -P -D foo -d -df -- ./a*, although the latter form is considered canonical.

> We strictly recommend to use long options. At first some options are not availabe in the short form and second the validation and understanding is much harder if the short option is used. Instead use some wrapper functions like described in the example section below.

With long options, options with mandatory arguments may be provided with an equal sign ('=') in a single argument, or as two arguments, and options with optional arguments must be provided as a single argument. In other words, *rjungle --def foo --debug a* is equivalent to *rjungle --define=foo --debug= -- ./a*, although the latter form is considered canonical (not to mention more robust, in case a future version of Random Jungle introduces an option named '--default').

Random Jungle understands the following options, grouped by functionality.

2.1 Command line options for operation modes

Several options control the overall operation of rjungle:

`-h`

`--help` Print a help summary on standard output, then immediately exit rjungle without reading any input files or performing any other actions.

`-Z`

`--version`
: Print the version number of the program on standard output, then immediately exit `rjungle` without reading any input files or performing any other actions.

`-f` *FILENAME*
`--file=`*FILENAME*
: *FILENAME* of input file with data. Input data has to been numerical. The default *FILENAME* is input.dat(.gz). In R (http://www.r-project.org), save data to file using `write.table` as follows:

```
> make.your.data.in.R
> write.table(yourData, file = "input.dat", row.names = FALSE,
              quote = FALSE)
> quit()
$ rjungle -f input.dat [...]
```

In plink (pngu.mgh.harvard.edu/~purcell/plink/), save data to raw file using the recodeA option, set the ped file option and char memory mode option in `rjungle` as follows:

```
$ plink --file yourDataFile --recodeA
$ rjungle -f yourDataFile.raw -p -M 2 [...]
```

Avoid missing values in data. See See Chapter 3 [Input data], page 11, for more details.

`-o` *FILEPREFIXNAME*
`--outprefix=`*FILEPREFIXNAME*
: *FILEPREFIXNAME* of output files is the first part of output files (i.e. `rjungle.importance`, `rjungle.prediction`, ...). The default *FILEPREFIXNAME* is `rjungle`. Use for example `-o my_analysis_no123`.

`-e` *CHAR*
`--delimeter=`*CHAR*
: Set the delimeter in your input file to *CHAR*. Default is a space.

`-w`
`--write=`*ID*
: Save Random Jungle model for a later prediction by `-P`.

`=` *0* [*ID*]
: not

`=` *1* [*ID*]
: to a gzipped XML file *FILEPREFIXNAME*.jungle.xml.gz (Not supported yet)

`=` *2* [*ID*]
: to a XML raw file *FILEPREFIXNAME*.jungle.xml

`=` *3* [*ID*]
: probability estimationt o a XML raw file *FILEPREFIXNAME*.jungle.xml

DEFAULT is 0.

`-P` *FILENAME*
`--predict=`*FILENAME*
: Predict test data with model (forest) saved by *-w2* given by the file *FILENAME* (test data file is given with option `-f`). Predictions will be written to *FILEPREFIXNAME*`.prediction`.

`--probability`
: Writes probability predictions to file *FILEPREFIXNAME*`.prediction`. Choose *-w3* for the training of the model!

`--deterministic`
: Starts a deterministic forest with variables specified in *-C*.

  FILEPREFIXNAME`.prediction`. Choose *-w3* for the training of the model!

`-y` *ID*
`--treetype=`*ID*
: Choose base classifier by setting *ID* There are serveral treetypes but only CART is fully supported. Explanatory or exposure variables will be named: input variables. Explained or response variable will be named: output variable. If you want to use CART like Random Forest (TM) does choose one of three possible values as follows:

  `=` *1 or 5* [*ID*]
  : CART, y (output variable): nominal, x (input variable(s)): numeric, *ID* = 1 is recommended for less different values in the input variables (i.e. GWA SNP data or integer data). *ID* = 5 is recommended for more different values in the input variables (i.e. many floating point numbers). Like original Breiman/Cutler/Friedman algorithm.

  `=` *2* [*ID*]
  : CART, y (output variable): nominal, x (input variable(s)): nominal,

  `=` *3* [*ID*]
  : CART regression trees, y (output variable): numeric, x (input variable(s)): numeric,

  `=` *4* [*ID*]
  : CART regression trees, CART regression trees, y (output variable): numeric, x (input variable(s)): nominal,

  DEFAULT is 1.

`-t` *SIZE*
`--ntree=`*SIZE*
: *SIZE* is the number of trees in jungle. If *SIZE*=0 then the size will be set automatically depending on mtry and variable size (experimental feature). DEFAULT is 500.

`-m` *SIZE*
`--mtry=`*SIZE*
: *SIZE* of randomly choosen variable sets. At each node building step, a variable will be selected out of the set, that serves the biggest information gain. The

bigger *SIZE* is set, the higher computing time might be. The bigger *SIZE* is set, the more similar trees in jungle will be. High noised data sets should processed with a big *SIZE*. Default is square root of number of input variables.

`-x` *NUM*
`--missingcode=`*NUM*
: Missings should always coded as `NA` or *NUM* in your data. The program takes *NUM* as a internal representation of a missing value. DEFAULT: -99.

`-i` *ID*
`--impmeasure=`*ID*
: Variable selection: Choose an method for estimating variable importance as follows:

  `=` *1* [*ID*]
  : Intrinsic Importrance (i.e. GINI-Index).

  `=` *2* [*ID*]
  : Permutation Importance by Breiman, Cutler (observed in Fortran code).

  `=` *3* [*ID*]
  : Permutation Importance by Liaw, Wiener (in R-package RandomForest).

  `=` *4* [*ID*]
  : Permutation Importance, raw values, no normalization.

  `=` *5* [*ID*]
  : Permutation Importance by Meng et. al

  The results will be written to file *FILEPREFIXNAME*`.importance`. You can not turn off variable importance output. DEFAULT is 1.

`-B` *ID*
: Variable selection/ model optimization: Choose an method for estimating variable importance as follows:

  `=` *0* [*ID*]
  : No backward elimination.

  `=` *1* [*ID*]
  : Backward elimination. Discard 50% at each step (slow). Stop if number of variables shrinked to *STOPSIZE*, see option `-j`.

  `=` *2* [*ID*]
  : Backward elimination. Discard 33% at each step (slow). Stop if number of variables shrinked to *STOPSIZE*, see option `-j`.

  `=` *3* [*ID*]
  : Backward elimination. Discard only negative values at each step (slow/recommended). (Shown at IGES2007 by Inke R. Konig).

  DEFAULT is 0.

`-j` *STOPSIZE*
`--nimpvar=`*STOPSIZE*
: Only necessary if `--impmeasure` = 2,3,5,6 or 7. How many variable should remain. The lesser *STOPSIZE* is, the reliable the result might be. The smaller SIZE is, the higher computing time will be. DEFAULT is 100.

`-v`
`--verbose`
: Print some nice information to screen. Otherwise put information to file *FILEPREFIXNAME*`.verbose` (DEFAULT).

`-u`
`--downsampling`
: Choose randomly samples without replacement. DEFAULT: switched off.

`-M` *ID*
`--memmode=`*ID*
: Usage of the heap memory (RAM) as follows:

  `=` *0* [*ID*]
  : Double precision floating point (BIG).

  `=` *1* [*ID*]
  : Single precision floating point (Normal).

  `=` *2* [*ID*]
  : Char (small). CHAR normally fits in one byte. DATA CELL VALUE HAS TO BE AN INTEGER IN [-127..127].

  If you want to use very small data coding, i.e. for SNP analysis, give `rjunglesparse` a try! DEFAULT is 0.

`-C` *FILE*
`--colselection=`*FILE*
: Only use selected columns listed in *FILE*. Example content of *FILE*:

  ```
  var1
  var20
  var1000
  var300
  ```

  DEFAULT is take all variables.

`-D` *NAME*
`--depvarname=`*NAME*
: Output variable name in the data SET! If *NAME* is empty then the `rjungle` switches to unsupervised mode.

`-s`
`--sampleproximities`
: It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data. Can be used as the distance matrix for Multidimensional Scaling (MDS). The results will be written to file *FILEPREFIXNAME*`.samproximity`. DEFAULT: switched off

`-z`
`--seeed` Seed of random number generators.

`-U`
`--nthreads=`*NUM*
Maximally use *NUM* threads (CPUs) for parallel processing. Limit for *NUM* is number of CPUs in computer. DEFAULT: Number of CPUs in computer.

`-p`
`--pedfile`
Input file has got ped format (i.e. plink output with recodeA). DEFAULT: switched off.

`-I` *NUM*
`--impute=`*NUM*
Impute missings in input data using Random Forest(TM)'s imputation algorithm. The number of iterations is given by *NUM*. For imputing continuous data, use option `-A` (`--impcont`) as well. For more information, have a look at http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm. Do not try to impute untyped SNPs (Schwarz et al. 2009, BMC Proceedings, 3, S65) if case-control-status is missing. Try a different program like: IMPUTE, MACH, PLINK, ... DEFAULT: switched off.

`-k` *NUM*
`--maxtreedepth=`*NUM*
This is a stop criterium/tunning parameter. Tree growing will stop, when the tree exceeds a depth of *NUM*. DEFAULT: switched off.

`-l` *NUM*
`--targetpartitionsize=`*NUM*
This is a stop criterium/tunning parameter. Tree growing will stop, when a partion falls below a size of *NUM* samples. DEFAULT: switched off.

`-K` *NUM*
`--condimp=`*NUM*
Perform conditional importance if option -i > 1. NUM is the pearson's cor. coef. cutoff. The smaller NUM, the bigger a conditional importance permutation group will be created. (=> More accurate, but slower) Requires: 0 <= NUM <= 1 NUM < 0 => switched off DEFAULT: switched off.

`-W`
Adjust class weights of unbalanced datasets automatically. DEFAULT: switched off.

`--oobset` Outputs the oobset of forest in file *.oob (each row == one tree) DEFAULT: switched off.

`--classweights=`*STRING*
Sets the weights of classes E.G.: "1.23;22;" ... first class gets weight 1.23 ... second class gets weight 22 DEFAULT: switched off

2.2 Experimental command line options for operation modes

The following options are more or less experimental without a deep validation process. Be cautious about using one of those.

`-d -g -n -r -c -a -b -w -V -S -A -G -q -E -X -Q`
: Those options are experimental. Be cautious about using one of those.

`-d`
`--depvar=`*`POS`*
: Output variable at column POS in the data SET. POS=[0-...] DEFAULT is 0 and switched off.

`-g`
`--depvarcol=`*`POS`*
: Output variable is the variable POS in the data FILE. POS=[0-...] DEFAULT is 0 and switched off.

`-n`
`--varnamesrow=`*`POS`*
: Variable names at row POS in data set. POS=[0-...] DEFAULT is 0 and switched off.

`-r`
`--nrow=`*`SIZE`*
: Number (SIZE) of samples. SIZE=[1-...]. SIZE = 0: use all samples in data file DEFAULT is 0.

`-c`
`--ncol=`*`SIZE`*
: Number (SIZE) of input variables (features). SIZE=[1-...]. SIZE = 0: use all variables in data file DEFAULT is 0.

`-a`
`--skiprow=`*`SIZE`*
: Skip SIZE rows (samples) before reading data from file. SIZE=[1-...] DEFAULT is 0.

`-b`
`--skipcol=`*`SIZE`*
: Skip SIZE columns (samples) before reading data from file. SIZE=[1-...] DEFAULT is 0.

`-S`
`--summary`
: Print a summary of all trees in jungle. DEFAULT: switched off.

`-A`
`--impcont`
: Impute data is continuous. DEFAULT: switched off (categorical data).

`-G`
`--gwa` Impute data with GWA settings. DEFAULT: switched off.

`-q`
`--tunemtry=`*THETA*
: Tune mtry parameter. Step size = THETA. E.g.: Quarter stepping (-q 0.25): 100 predictor variables (ncol = 100) => mtry = 25, 50, 75. DEFAULT: switched off.

`-E`
`--extractdata`
: Extract data to file. DEFAULT: switched off.

`-X`
`--plugin=`*FILE*
: Use Random Jungle generator from given plugin: *FILE*. Totally experimental. Please contact the maintainer for more and detailed information on the plugin process. So far this part is not fully supported. DEFAULT: switched off (Empty String).

`-Q`
`--pluginpar=`*STRING*
: Additional plugin parameters. As decribed above this part is totally experimental and not under maintance. DEFAULT: switched off (Empty String).

3 Input data

This chapter describes various input file types of `rjungle`.

> Remember that the Random Forest (TM) approach can not handle missing values. *Random Jungle* has a implemented imputation algorithm but it should be used carefully. Please consider well known literature for your special problem or remove all missings from the data (not recommended). We demonstrate some wrapper functions in the example section below.

3.1 Input the whole data

`Random Jungle` analyses data given in a file and name of response variable, see option `-f` and `-D`. The file format is matrix like and is as follows:

`names` [Variable]
: The first line of the input file contains the variable names. The variable name must not contain quotes or space charaters. The variables are seperated by space charaters.

`cells` [Data]
: Each following line represents one sample (observation). Every single sample needs to have one numerical value for each variable. The values are seperated by space charaters and are ordered corresponding to the variable names (of course).

Here, an example of an input file `indat.data`:

```
responseVar inputVar1 inputVar2 inputVar3 inputVar4
0 1.2 3.4 5.6 7.8
0 1.1 3.3 5.5 7.7
0 2.2 4.4 6.6 8.8
1 1.0 3.0 5.0 7.0
1 1.0 3.0 5.0 7.0
1 2.0 4.0 6.0 8.0
```

The corresponding call would be $ *rjungle --file indat.data -D responseVar [...]*

If you choose the ped file option `-p` then the input file format is ped like. The file must have variables FID, IID, PAT, MAT, SEX, PHENOTYPE and at least three variables (SNPs). Here, an example of an input file:

```
FID IID PAT MAT SEX PHENOTYPE rs1 rs2 rs3 rs4
1000 NA2001 0 0 2 0 0 1 0 1
1000 NA2002 0 0 2 0 0 2 0 1
1001 NA2003 0 0 1 0 1 2 0 1
1001 NA2004 0 0 1 0 1 1 1 1
1002 NA2005 0 0 2 1 1 1 1 1
1004 NA2006 0 0 2 1 1 2 1 1
1005 NA2007 0 0 2 1 2 2 2 2
1006 NA2008 0 0 1 1 2 2 2 2
1007 NA2008 0 0 1 1 2 2 2 2
```

3.2 Restricted analysis

The `rjungle` can be run also with just a subset of all variables which are given in the input file (option `-f`). See description of option `-C`, for more details.

4 Output data

This chapter describes various output file types of `rjungle`.

4.1 The standard output files

If `rjungle` is executed it will always produce three files: Log file, Confusion file, and Importance file. You can run an example in your `randomjungle-2.0.0` directory and see it.

```
$ rjungle -f your_data_file.txt -D response_variable_name -v
$ ls
rjungle.importance  rjungle.log  rjungle.prediction  test.ped
```

Or without PED file as follows:

```
$ rjungle -f your_data_file.txt -D response_variable_name -v
```

4.2 Log file

The log file contains all options / parameter of the `rjungle` execution. This is useful to reproduce results.

4.3 Confusion file

The `rjungle` always evaluates the classifier. The accuracy on training and test data. Test data is called oob data which is collected during growing process (similar to cross validation's test sets). The results are shown in so-called confusion matrices in file *FILEPREFIXNAME*`.confusion`. Columns represent the predicted values and rows represent the real values.

When using option `-y1` a file *FILEPREFIXNAME*`.confusion2` will be created. The file contains class specific error rates.

4.4 Prediction file

The `rjungle` is able to perform new data to a saved classifier (option `-w2` and `-P`). The predicted results will be saved to file *FILEPREFIXNAME*`.prediction`.

4.5 Importance file

`Random Jungle` estimates the importance of variables (option `-i`). The results are saved to file *FILEPREFIXNAME*`.importance` and/or *FILEPREFIXNAME*`.importance2`. The first file contains four columns but most interesting columns are `varname` (variable name) and `value` (importance value). The higher the `value`, the more important is the variable with name `varname`. This list is sorted ascending. So, look at the tail of file to see the most important variables. The second file contains various permutation importance informations.

4.6 Verbose file

We recommend to use always `-v` in each `Random Jungle` run. Especially new users should use this option to see what `Random Jungle` is doing and if all wanted options are active. In addition a timer is given with a processing information.

If `Random Jungle` is invoked from command line without verbose option it is very schtum and do not output anything to screen. It writes all information occuring during a run to file *FILEPREFIXNAME*`.verbose`. Nevertheless, if you want a takly `rjungle` which puts all process information to screen then use option (`-v`)

```
$ rjungle -f your_data_file.txt -D PHENOTYPE -v

Start: ...
+---------------------+-----------------+-------------------+
|    RandomJungle     |      ......     |      ......       |
+---------------------+-----------------+-------------------+
|                             ......                        |
+---------------------------------------------------------+

Output to:
rjungle.*
loading data...
Read 9 row(s) and 10 column(s).
Use 9 row(s) and 6 column(s).
dependent variable name: PHENOTYPE
Growing jungle...
Number of variables: 6 mtry = 2
1 thread(s) growing 500 tree(s)
Growing time estimate: ~0 sec.
Generating and collecting output data...
Compiling trees.
Writing accuracy information...
calculating confusion matrix...

Elapsed time: 0 sec
Finished: ...

$
```

5 Some nice examples

This chapter describes various examples of working with `rjungle`. We concentrate here on examples included into the R project (http://www.r-project.org/) and connected packages. Why do we use `Random Jungle` invoking R? First, a normal project has some data handling issues. It is much more easier to do with R, like impute missing values or renaming and removing columns. Second, if the `Random Jungle` code is included into a R script it can be validated and easy repeated.

The following example code is written in R. Therefore, we skiped the `>` in each line for a better copy&paste behaviour. R output instead is marked with an `>`. Shell commands are still marked by `$`.

5.1 Rjungle in R

First of all the general framework for `Random Jungle` in R. We build up `"paste"` one command string `rjungleCMD` and send it to the system e.g. shell.

```
## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleInFile <- file.path("/to/inDir/dataWithoutMissings.dat")
rjungleOutFile <- file.path("/to/outDir/projectname")

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleInFile,
                    "-v",                  ## show processing
                    "-D responseVar",      ## response variable name
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))                    ## send to system
```

Question: Where do I get the `"rjungleExe"`? Visit the project webpage of `Random Jungle` (http://www.r-project.org/) and download the pre-compiled version of `Random Jungle` matching to your system.

Question: What mean `"responseVar"`? This is your variable of interest. Like the diabtes state or written in R formula: `responseVar ~ Var1 + Var2`.

5.2 Simple example in R

We want to grow a simple forest on the `iris` data. So, our input data contains 4 variables (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`) and 1 response variable(`Species`). Caution, we write all results files into the working directionary of R! Change the `rjungleOutFile` for a better data handling.

```
## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleInFile <- file.path("iris.dat")
rjungleOutFile <- file.path("iris")

## Data handling
```

```
data(iris)
iris$Species = as.integer(iris$Species)   ## convert factor to integer
write.table(iris, file = rjungleInFile, row.names = FALSE, quote =
FALSE)

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleInFile,
                    "-v",                 ## show processing
                    "-D Species",         ## response variable name
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))                   ## send to system
```

Question: Where do I find the results? The results are written into the directionary of R or the specification of `rjungleOutFile`. Therefore, where your R script is running now. To get this information type `getwd()`.

5.3 Simple example with ped file

Using Plink PED-files:

```
## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", pedFile,
                    "-v",                 ## show processing
                    "-p",                 ## read in pedFile
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))                   ## send to system
```

Question: What are PED-files? Visit http://pngu.mgh.harvard.edu/~purcell/plink/ for more information.

Question: I have build up an PED-file but it does not work! Have you used the `recodeA` option? This is important for `Random Jungle` to handle PED-Files.

Question: It takes hours to analyse my data! Look at the sparse version of `Random Jungle` and other examples below.

5.4 Estimating variable importance

In the second example we used only a small data set. Know we want to analyse the Pima Indian Diabetes data set. In R the data is available in the package `MASS`. Firstly, lets estimate the fast GINI-Importance (`i1`) with 1000 trees.

```
library(MASS)

## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleInFile <- file.path("pima.dat")
rjungleOutFile <- file.path("pima")

## Data handling
```

```
data(Pima.tr)
sum(is.na(Pima.tr))    ## Is there any missing?
write.table(Pima.tr, file = rjungleInFile, row.names = FALSE, quote =
FALSE)

## Run Random Jungle
rjungleGiniCMD <- paste(rjungleExe,
                        "-f", rjungleInFile,
                        "-v",                ## show processing
                        "-i1",               ## chose GINI-Importance
                        "-t1000",            ## 1000 trees
                        "-D type",           ## response variable name
                        "-o", rjungleOutFile) ## out file path
try(system(rjungleGiniCMD))
```

Secondly, we now want to estimate the permutation importance (`i2`).

```
## Run Random Jungle
rjunglePermCMD <- paste(rjungleExe,
                        "-f", rjungleInFile,
                        "-v",                ## show processing
                        "-i2",               ## chose permutation-Importance
                        "-t1000",            ## 1000 trees
                        "-D type",           ## response variable name
                        "-o", rjungleOutFile) ## out file path
try(system(rjunglePermCMD))
```

Now, we got even a more reliable result.

Finally, we want to use the conditional variable importance (`--condimp`). Caution it is much more slower than the permutation importance alone!

```
## Run Random Jungle
rjunglePermCMD <- paste(rjungleExe,
                        "-f", rjungleInFile,
                        "-v",                ## show processing
                        "-i3",               ## chose permutation-Importance
                        "--condImp 0.2",     ## run with condImp of 0.2
                        "-t1000",            ## 1000 trees
                        "-D type",           ## response variable name
                        "-o", rjungleOutFile) ## out file path
try(system(rjunglePermCMD))
```

Now we can read in the wanted importance file.

```
## Read in the importance file
rjungleGiniImportanceFile <- paste(rjungleOutFile, "importance", sep = "")
rjunglePermImportanceFile <- paste(rjungleOutFile, "importance2", sep = "")
importanceGini <- read.table(rjungleGiniImportanceFile)
importancePerm <- read.table(rjunglePermImportanceFile)
```

Question: There are many importance values. Which one should I use? We recommend to use the Liaw score which is the same as in the R package `randomForest`. Again, look at

the options list (`-i`) and get more information on the used importance measures. Depending on your problem a different important measure might be feasible.

5.5 Using R and rjungle seperately

We want to analyse data from R and do not want to start `Random Jungle` out of R. Instead we prepare the data in R and start `Random Jungle` from the shell.

```
$ R
...
data(iris)
iris$Species = as.integer(iris$Species) # convert factor to integer
write.table(iris, file = "iris.dat", row.names = FALSE, quote = FALSE)
quit("no")
$ rjungle -f iris.dat -v -D Species -o iris
$ cat iris.confusion
...
$ cat iris.importance
...
```

It is not recommended to use R and `Random Jungle` in this way.

5.6 Using plink and rjungle

We assume you have your prepared GWA data called `gwadata`.

```
## File handling for PLINK
plinkExe <- file.path("/to/executable/")
plinkInFile <- file.path("gwadata")

## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleInFile <- file.path("pima.dat")
rjungleOutFile <- file.path("pima")

## Run PLINK
plinkCMD <- paste(plinkExe,
                  "--file", plinkInFile,
                  "--recodeA")
try(system(plinkCMD))

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleInFile,
                    "-v",                 ## show processing
                    "-i1",                ## chose GINI-Importance
                    "-p",                 ## use PED-file
                    "-M2",                ## memory mode
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))
```

```
## Read in the importance file
rjungleImportanceFile <- paste(rjungleOutFile, "importance", sep = "")
importance <- read.table(rjungleImportanceFile)
```

or

```
rjungleExe <- file.path("/to/executable/rjunglesparse")

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleInFile,
                    "-v",                 ## show processing
                    "-i1",                ## chose GINI-Importance
                    "-p",                 ## use PED-file
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))

## Read in the importance file
rjungleImportanceFile <- paste(rjungleOutFile, "importance", sep = "")
importance <- read.table(rjungleImportanceFile)
```

The `rjunglesparse` uses less memory than `rjungle`. But your data should only use the values 0,1,2 and 3(missing code).

Question: What is PLINK? Visit http://pngu.mgh.harvard.edu/~purcell/plink/ for more information.

Question: Where is the difference between both Rjungle runs? In the second one we use `rjunglesparse`.

5.7 Prediction (classification)

See examples above for the generation of the `iris.dat` data file.

```
## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleTrainFile <- file.path("iris.dat")
rjungleOutFile <- file.path("iris")

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleTrainFile,
                    "-w2",                ## save trees for prediction
                    "-v",                 ## show processing
                    "-D Species",         ## response variable name
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))

## Run prediction
rjungleTestFile <- file.path("iris.dat")
rjungleXmlFile <- file.path("iris.jungle.xml")
```

```
rjunglePredCMD <- paste(rjungleExe,
                        "-f", rungleTestFile,
                        "-P", rjungleXmlFile, ## load saved trees for prediction
                        "-v",                 ## show processing
                        "-D Species",         ## response variable name
                        "-o", rjungleOutFile) ## out file path
try(system(rjunglePredCMD))

## Get prediction
rjunglePredFile <- file.path("iris.prediction")
predRjungle <- read.table(rjunglePredFile)
```

Question: Is it not bad to use the same data for training and testing? Yes. You should avoid this. This example is only for demonstration.

5.8 Prediction (probability)

Here we present the probability estimation using `Random Jungle`. Please consider the project page for a overview of literature dealing with this issue (http://www.randomjungle.de/).

```
## File handling for Random Jungle
rjungleExe <- file.path("/to/executable/rjungle")
rjungleTrainFile <- file.path("iris.dat")
rjungleOutFile <- file.path("iris")

## Run Random Jungle
rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleTrainFile,
                    "-w3",                ## save trees for prediction
                    "-v",                 ## show processing
                    "-D Species",         ## response variable name
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))

## Run prediction
rjungleTestFile <- file.path("iris.dat")
rjungleXmlFile <- file.path("iris.jungle.xml")
rjunglePredCMD <- paste(rjungleExe,
                        "-f", rungleTestFile,
                        "--probability",
                        "-P", rjungleXmlFile, ## load saved trees for prediction
                        "-v",                 ## show processing
                        "-D Species",         ## response variable name
                        "-o", rjungleOutFile) ## out file path
try(system(rjunglePredCMD))

## Get prediction
rjunglePredFile <- file.path("iris.prediction")
```

```
predRjungle <- read.table(rjunglePredFile)
```

Question: What is the difference to the classification prediction? I do not see it! You save the trees in a different format by using the option `-w3` in the training step and predict the probabilties by using `--probability`.

Question: What is the short option for `--probability`? There is no short option for `--probability`.

Question: Is it not bad to use the same data for training and testing? Yes. You should avoid this. This example is only for demonstration.

5.9 Deterministic Forest

If you want to force `Random Jungle` to select at each split **always** some variables than run a deterministic forest. In this example we would choose `age` and `sex` at each split among the other random selected variables in the data set.

```
wantedColumns <- c("age", "sex")
write.table(wantedColumns, file = "wantedColumns.txt",
            row.names = FALSE, col.names = FALSE, quote = FALSE)

rjungleCMD <- paste(rjungleExe,
                    "-f", rjungleInFile,
                    ...
                    "-deterministic",
                    "-C", wantedColumns.txt,
                    ...
                    "-o", rjungleOutFile) ## out file path
try(system(rjungleCMD))
```

5.10 Imputation

The following examples are describing the imputation by `Random Jungle`. Please be caution. You will manage to get rid of missings but it might procude false positives. Please consider the right literature matching to your problem to avoid missings. We cannot give you a general framework. Still, remove all missings, like `na.omit()`.

SNP data (raw/fast imputation):

```
$ rjunglesparse -f mypedfile.raw -p -t1 -I1 -o example8_1_1
```

The imputation result was written to example8_1_1.imputed.dat.gz and can be used for analysis. E.g.:

```
$ rjunglesparse -f example7_1_1.imputed.dat.gz -p ... -o example8_1_2
```

Imputing continuous data:

```
$ rjungle -f continuous.dat -A -D responseVar -I6 -o example8_2
```

or imputing categorical data:

```
$ rjungle -f cate.dat -D responseVar -I6 -o example8_3
```

or imputing data with no groups to classify on (unsupervised learning):

```
$ rjungle -f continuousAndNoGroups.dat -A -I5 -o example7_4
```

Question: I have SNP data are there other possibilities? Yes. Look for PLINK or IMPUTE.

5.11 Using plink and rjunglesparse

The `rjunglesparse` is the same program like `rjungle`, but you can use just a small set of values: 0,1,2 (and 3 as missing coding). Look at Section 5.6 [Example5], page 18 for a implementation in R. You might want to use `rjunglesparse` in conjunction with `plink`. The memory consumption of `rjunglesparse` is very small.

```
$ plink --file gwadata --recodeA
$ rjunglesparse -f plink.raw -p -v -o example8
...
$ tail example8.importance
```

5.12 Using MPI

For high speed parallel processing, `Random Jungle` could be used on computer clusters using MPI mode **(`Random Jungle` has to be compiled for MPI!)**. The program was performed on huge data successfully using 150 processors (300 processes) in parallel on a high performance cluster. However, the MPI mode should be used by experienced users only. In MPI mode, performing permutation importance (option `-i2`, .., `-i5`) is allow exclusivly. Execute `Random Jungle` as follows (one possibility, look at your job system, it might differ):

```
$ plink --file gwadata --recodeA
$ mpirun -np 200 --host hostname1, hostname2, ...
         rjunglesparse -f plink.raw -p -v -i4 -o example9
...
$ tail example9.importance
```

Each process writes temp files to working directory (`example9_mpi_id_*.*`). Final results are written to usual files (`example9.*`).

Question: Where do I get the MPI pre-compiled versions? Please look at the project homepage (`http://www.randomjungle.de/`). If there is no MPI pre-compiled version, it is not possible at the moment.

Question: MPI does not work on the cluster etc.? Please contact your admin. Depending on your batch or job system there are many sources of errors.

Appendix A Indices of concepts and macros

A.1 Compiling and installing Random Jungle

There are pre-compiled versions on the internet site http://www.randomjungle.de/. Maybe, there exists a pre-compiled version for your platform (computer) and you do not need to compile it.

> The source code is not freely available. Please write to infoimbs@imbs-luebeck.de and ask for a possible cooperation or a pre-compiled version for your system.

Random Jungle uses some GNU tools, libraries and other free open source code. The installation routine (./configure) prompt you to install packages which are missing on your machine. It is also recommended to compile the sources with an openMP supporting compiler (i.e. GNU gcc > 4.2). Invoke the following commands to compile rjungle:

```
./configure
make

make check
make install
```

For compiling Random Jungle with MPI (message passing interface) support, configure RJ using option --enable-mpi. The program is optimized for Open MPI (http://www.open-mpi.org/). The MPI mode should be used by experienced users only.

```
./configure --enable-mpi
```

A.2 Index for many concepts