# Quick Start Guide for **wsrf**

He Zhao

May 30, 2014

## 1 Introduction

The R package **wsrf** is a parallel implementation of the Weighted Subspace Random Forest algorithm (wsrf) of [4]. A novel variable weighting method is used for variable subspace selection in place of the traditional approach of random variable sampling. This new approach is particularly useful in building models for high dimensional data—often consisting of thousands of variables. Parallel computation is used to take advantage of multi-core machines and clusters of machines to build random forest models from high dimensional data with reduced elapsed times.

## 2 Requirements and Installation Notes

Currently, **wsrf** requires R (>= 3.0.0), Rcpp (>= 0.10.2). For the use of multi-threading, a C++ compiler with C++11 standard support of threads or the Boost C++ library [1] with version above 1.54 is required. The choice is available at installation time depending on what is available to the user. To install the latest version of the package, from within R run:

```
install.packages("wsrf")
```

By default, multi-threading functionality is not enabled, which can be configured through the argument `configure.args`.

```
install.packages("wsrf", configure.args="--enable-c11=yes")
```

We recommend using C++11 standard library for accessing multi-threaded functionality, which will be our main focus for development in the future. Though support for compiling C++11 code in packages is not available in current release of R, it has been tested that it can be compiled if the user has already installed the latest version of GCC and C++ standard library[1].

Besides the default installation for C++11, we also provide another implementations of **wsrf**, which implements parallelism using Boost.

The choice of version to install is available at build time. The version without parallelism, as required when C++11 is not available nor is Boost, and is the recommended and only possible choice for Microsoft Windows platform with the current version of R (3.0) (the same as the first installation method above):

[1] C++11 support is experimental in R-devel now, see `http://developer.r-project.org/blosxom.cgi/R-devel/NEWS/2013/12/02#n2013-12-02`

```
install.packages("wsrf",
                 configure.args="--enable-c11=no")
```

Finally the version using Boost for multithreading can be installed with the appropriate configuration options. This is suitable when the version of C++ available does not support C++11.

```
install.packages("wsrf",
                 configure.args="--with-boost-include=<Boost include path>
                                 --with-boost-lib=<Boost lib path>")
```

## 3 Usage

This section demonstrates how to use **wsrf**, especially on a cluster of machines.

The example uses a small dataset *weather* from **rattle** [3]. See the help page of **rattle** in R (`?weather`) for more details of *weather*. Below are the basic information of it.

```
library(rattle)
ds <- weather
dim(ds)

## [1] 366  24

names(ds)

##  [1] "Date"          "Location"      "MinTemp"       "MaxTemp"
##  [5] "Rainfall"      "Evaporation"   "Sunshine"      "WindGustDir"
##  [9] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
....
```

Before building the model we need to prepare the training dataset. First we note the various roles played by the different variables, including identifying the irrelevant variables

```
target <- "RainTomorrow"
id     <- c("Date", "Location")
risk   <- "RISK_MM"
ignore <- c(id, if (exists("risk")) risk)

(vars <- setdiff(names(ds), ignore))

##  [1] "MinTemp"       "MaxTemp"       "Rainfall"      "Evaporation"
##  [5] "Sunshine"      "WindGustDir"   "WindGustSpeed" "WindDir9am"
##  [9] "WindDir3pm"    "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm"   "Pressure9am"   "Pressure3pm"   "Cloud9am"
....
```

```
dim(ds[vars])

## [1] 366  21
```

Next we deal with missing values, using `na.roughfix()` from **randomForest** to take care of them.

```
library(randomForest)
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])
ds[target] <- as.factor(ds[[target]])
(tt <- table(ds[target]))

##
##  No Yes
## 300  66
```

We construct the formula that describes the model which will predict the target based on all other variables.

```
(form <- as.formula(paste(target, "~ .")))

## RainTomorrow ~ .
```

Finally we create the randomly selected training and test datasets, setting a seed so that the results can be exactly replicated.

```
seed <- 42
set.seed(seed)
length(train <- sample(nrow(ds), 0.7*nrow(ds)))

## [1] 256

length(test <- setdiff(seq_len(nrow(ds)), train))

## [1] 110
```

The signature of the function to build a weighted random forest model in **wsrf** is:

```
wsrf(formula,
     data,
     ntrees=500,
     nvars=NULL,
     weights=TRUE,
     parallel=TRUE)
```

We use the training dataset to build a random forest model. All parameters, except "formula" and "data", use their default values: 500 for "ntrees" — the number of trees, the same as other package (**randomForest** and **party**); TRUE for "weights" — weighted subspace random forest or random forest; TRUE for "parallel" — use multi-thread or other options, etc.

```
library(wsrf)
model.wsrf <- wsrf(form, data=ds[train, vars])
print(model.wsrf, summary=TRUE)

##
## Tree 1 with 51 nodes:
## 1) root
##  ..2) Pressure9am <= 1013.2
....
```

Here in the output, *Strength* and *Correlation* are two measures introduced in [2] for evaluating a random forest model. *Strength* measures the collective performance of individual trees in a random forest and *Correlation* measures the diversity of the trees.

We can also obtain *Strength* and *Correlation* by:

```
strength(model.wsrf)

## strength
##   0.6213

correlation(model.wsrf)

## correlation
##      0.1866
```

Then, `predict` the classes of test data.

```
cl <- predict(model.wsrf, newdata=ds[test, vars], type="class")
actual <- ds[test, target]
(accuracy.wsrf <- sum(cl == actual, na.rm=TRUE)/length(actual))

## [1] 0.8455
```

Thus, we have built a model that is 84.5455% accurate on unseen testing data.

To compare with **cforest** and **randomForest**,

```
library(randomForest)
library(party)
model.randomForest <- randomForest(form, data=ds[train, vars])
model.cforest <- cforest(form, data=ds[train, vars])

cl <- predict(model.randomForest, newdata=ds[test, vars], type="response")
actual <- ds[test, target]
(accuracy.randomForest <- sum(cl == actual, na.rm=TRUE)/length(actual))

## [1] 0.8818
```

```
cl <- predict(model.cforest, newdata=ds[test, vars], type="response")
actual <- ds[test, target]
(accuracy.cforest <- sum(cl == actual, na.rm=TRUE)/length(actual))

## [1] 0.8091
```

Next, we will specify building the model on a cluster of servers.

```
servers <- paste0("node", 31:40)
model.wsrf <- wsrf(form, data=ds[train, vars], parallel=servers)
```

All we need is a character verctor specifying the hostnames of which nodes to use, or a named integer vector, whose values of the elements give how many threads to use for model building, in other words, how many trees built simultaneously. More detail descriptions about **wsrf** are presented in.

## References


[1] Boost Community. *Boost C++ Libraries*, 2013.

[2] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[3] Graham J. Williams. *Data Mining with* **Rattle** *and R: The art of excavating data for knowledge discovery.* Use R! Springer, 2011.

[4] Baoxun Xu, Joshua Zhexue Huang, Graham Williams, Qiang Wang, and Yunming Ye. Classifying very high-dimensional data with random forests built from small subspaces. *International Journal of Data Warehousing and Mining (IJDWM)*, 8(2):44–63, 2012.