randomForestSRC CHEAT SHEET

RFSRC
Fast Unified Forests

Basics

randomForestSRC is a fast OpenMP and memory efficient package for fitting random forests (RF) for univariate, multivariate, unsupervised, survival, competing risks, class imbalanced classification and quantile regression.
A basic grow call is of the form:

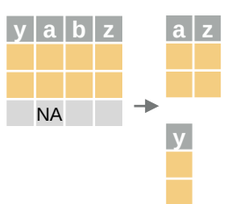
rfsrc(formula, data, ntree, mtry, nodesize)

Grow your RF through **rfsrc**,
specify your model in **formula**,
provide your data frame in **data**
and tune your model via **ntree, mtry, nodesize**.

Specify a formula

Survival	rfsrc(Surv(time, status) ~ ., data = veteran)
Competing Risk	rfsrc(Surv(time, status) ~ ., data = wihs)
Regression **Quantile Regression**	rfsrc(Ozone ~., data = airquality) quantreg(mpg ~ ., data = mtcars)
Classification **Imbalanced Two-Class**	rfsrc(Surv(time, status) ~ ., data=veteran) imbalanced(status ~ ., data = breast)
Multivariate Regression **Mixed Regression** **Quantile Regression** **MV Mixed Quantile**	rfsrc(Multivar(mpg, cyl) ~., data = mtcars) rfsrc(cbind(Species,Sepal.Length)~.,data=iris) quantreg(cbind(mpg, cyl) ~ ., data = mtcars) quantreg(cbind(Species,Sepal.Length)~.,data=iris)
Unsupervised **sidClustering** **Breiman (Shi-Horvath)**	rfsrc(data = mtcars) sidClustering(data = mtcars) sidClustering(data = mtcars, method = "sh")

Clean up and impute data



choose your variables in **formula** and grow a tree.

```
o <- rfsrc(y ~ a + z, data= dta, ntree = 1)
```

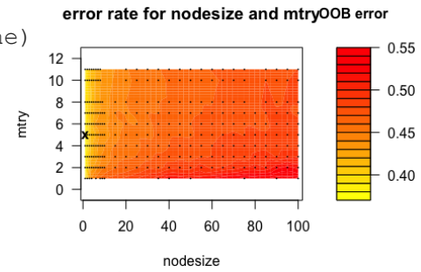
your outcome(s) will be saved in `o$y` and your predictors are in `o$x` from `dta` without missing values. To impute your data, use

```
o <- impute(y ~ a + z, data = dta)
o <- rfsrc(y ~ a + z, data = dta, na.action = "na.impute")
```

Tune mtry and nodesize

`tune` Find the optimal mtry and nodesize tuning parameter for a random forest using out-of-bag (OOB) error

```
o <- tune(quality ~ ., wine)
> o$optimal
nodesize      mtry
    1          5
```



`tune.nodesize` Find the optimal nodesize

Grow

Convenient interface for growing a CART tree

```
rfsrc.cart(formula, data, ntree = 1, mtry = ncol(data),
           bootstrap = "none")
```

Fast OpenMP parallel computing of random forests

```
rfsrc(formula, data, ntree = 500,
      mtry = NULL, ytry = NULL,
      nodesize = NULL, nodedepth = NULL,
      splitrule = NULL, nsplit = 10,
      importance = c(FALSE, TRUE, "none", "permute",
                     "random", "anti"),
      ensemble = c("all", "oob", "inbag"),
      bootstrap = c("by.root", "none", "by.user"),
      samptype = c("swor", "swr"),
      samp = NULL, membership = FALSE,
      na.action = c("na.omit", "na.impute"),
      nimpute = 1,
      ntime = 250, cause,
      proximity = FALSE, distance = FALSE,
      forest.wt = FALSE, xvar.wt = NULL,
      yvar.wt = NULL, split.wt = NULL,
      case.wt = NULL,
      forest = TRUE,
      var.used = c(FALSE, "all.trees", "by.tree"),
      split.depth = c(FALSE, "all.trees", "by.tree"),
      seed = NULL, do.trace = FALSE,
      statistics = FALSE, ...)
```

`rfsrc.fast` Fast approximate random forests using subsampling with forest options set to encourage computational speed

`rfsrc.anonymous` Random forests carefully modified so as not to save the original training data when sharing

`synthetic` Synthetic random forest using synthetic features

`imbalanced` Solutions to the two-class imbalanced problem

`quantreg` Univariate or multivariate quantile regression forest and returns its conditional quantile and density values

`sidClustering` Clustering of unsupervised data

Inference from the Forest

Ensemble Predicted Value for Training Data

```
o <- rfsrc(Ozone ~ ., data = airquality)
```

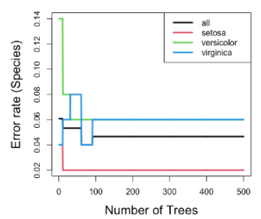
Inbag and out-of-bag (OOB) predicted values for the training dataset are in `o$predicted` and `o$predicted.oob`

Other Ensemble Values for Training Data

- For classification problem, we also have `$class` and `$class.oob` for class labels
- For survival problem, we have
  `$survival` and `$survival.oob` for survival function
  `$chf` and `$chf.oob` for cumulative hazard function
  `$cif` and `$cif.oob` for cumulative incidence function

Prediction Error for Assessing Model Performance

```
o <- rfsrc(Species ~ ., data=iris, block.size=1)
```



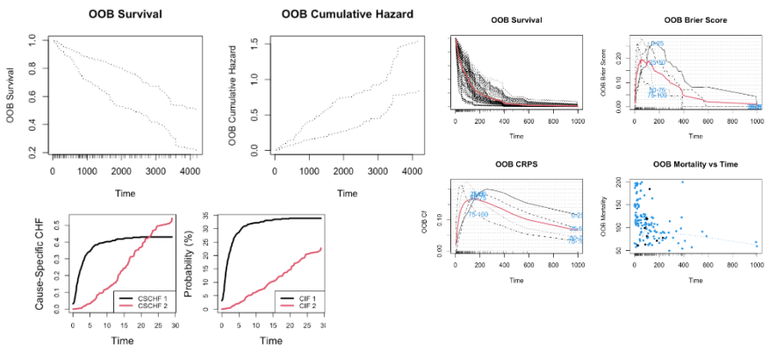
`o$err.rate` returns tree cumulative OOB error rate; `print(o)` lists OOB error rate in the bottom; `plot(o)` plots OOB error rate along with number of trees; `get.auc(y, prob)` obtains the value of AUC (area under the ROC curve)

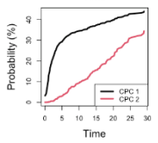
`get.mv.error` obtains error rate from a multivariate random forest

RFSRC
Fast Unified Forests

Visualization

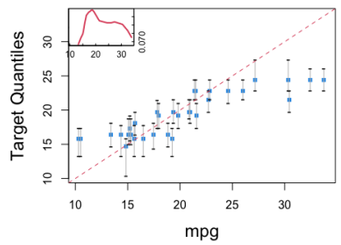
`plot.survival` plots various survival estimates





`plot.competing.risk` plots summary curves from a competing risk analysis

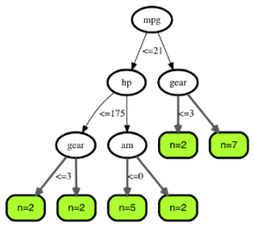
`plot.quantreg` plots quantiles obtained from a quantile regression forest



Tree Visualization

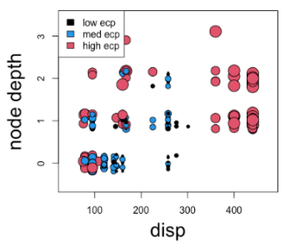
`get.tree` extract a single tree from a forest and plot it on your browser

```
mtcars.unspv <- rfsrc(data = mtcars)
plot(get.tree(mtcars.unspv, 5))
```



Split Statistics

`stat.split` acquires split statistic information. The end-cut preference (ECP) splitting property can be plotted



Predict on New Data

```
o.pred <- predict(object = o, newdata)
```

Predicted values for the new dataset are in `o.pred$predicted`

`get.mv.predicted` returns predicted value for multivariate regression analysis

Restore

Restoration using the `predict` function makes it possible for users to acquire information from the grow forest without the computational expense of having to regrow a new forest

Examples of restore are as follows (extract: proximity, variable splitting behavior, performance over specific trees)

```
o <- rfsrc(Ozone ~ ., data = airquality)

predict(o, proximity = TRUE)$proximity
predict(o.obj, var.used = "by.tree")$var.used
predict(o, get.tree=10:15)$err.rate
```

Variable Selection

Variable Importance (VIMP)

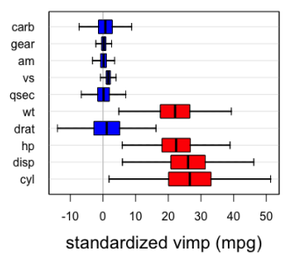
```
o <- rfsrc(Species ~ ., iris, importance = TRUE)
```
Or
```
obj <- rfsrc(Species ~ ., data = iris)
o <- vimp(obj)
```

`o$importance` returns permutation VIMP and `plot(o)` plots VIMP when setting importance to `"permute"` or `"TRUE"` in `rfsrc` or using `vimp`

`subsample` subsample forests for VIMP confidence intervals

`plot.sample` plots Subsampled VIMP confidence intervals

```
o <- rfsrc(mpg ~ ., mtcars)
smp.o <- subsample(reg.o, B=25,
                                subratio=.5)
plot.subsample(smp.o)
```



`holdout.vimp` calculates hold out VIMP from the error rate of blocks of trees grown with and without a variable

`get.mv.vimp` returns VIMP from a multivariate random forest

Minimal Depth

`max.subtree` extracts minimal depth and maximal subtree information used for variable selection and identifying interactions between variables

Variable Selection and Hunting

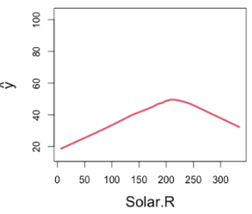
`var.select(formula, data, method)` Variable selection or hunting by setting method

`md`	Minimal depth (default)
`vh`	Variable hunting
`vh.vimp`	Variable hunting with VIMP

Partial Plot

Marginal Effect Plot

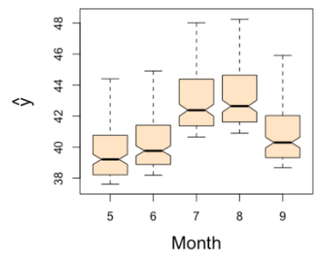
`plot.variable(o, xvar.names)`



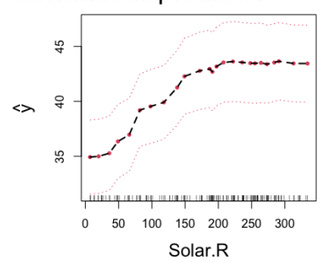
Partial Dependence Plot

`plot.variable(o, xvar.names, partial = TRUE)` and `partial`

Categorical predictor:



Continuous predictor:



Set `surv.type` for survival analysis:

`mort`	Mortality
`rel.freq`	Relative frequency of mortality
`surv`	Predicted survival, where the predicted survival is for the time point specified using *time*
`years.lost`	The expected number of life years lost
`cif`	The cumulative incidence function
`chf`	The cumulative hazard function

`get.partial.plot.data` is a handy function that parses the output from `"partial.rfsrc"` in format suitable for plots