randomForestSRC CHEAT SHEET

**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:

```
rfsr(formula, data, ntree, mtry, nodesize)
```
Grow your RF through `rfsr`, specify your model in `formula`, provide your data frame in `data` and tune your model via `ntree`, `mtry`, `nodesize`.

### Specify a formula

<table>
<thead>
<tr>
<th>Survival</th>
<th>rfsrc(Surv(time, status) ~ ., data = veteran)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Competing Risk</td>
<td>rfsrc(Surv(time, status) ~ ., data = whs)</td>
</tr>
<tr>
<td>Regression</td>
<td>rfsrc(Ozone ~ ., data = aqirquality)</td>
</tr>
<tr>
<td>Quantile Regression</td>
<td>quantreg(mpg ~ ., data = mtcars)</td>
</tr>
<tr>
<td>Classification</td>
<td>rfsrc(formula, data, status, ~ ., data=breast)</td>
</tr>
<tr>
<td>Imbalanced Two-Class</td>
<td>rfsrc(Multivar(mpg, cyl) ~ ., data = mtcars)</td>
</tr>
<tr>
<td>Mixed Regression</td>
<td>rfsrc(cb(min, Sepal.Length) ~ ., data=iris)</td>
</tr>
<tr>
<td>Quantile Regression</td>
<td>quantreg(cb(min, Sepal.Length) ~ ., data=iris)</td>
</tr>
<tr>
<td>MV Mixed Quantile</td>
<td>rfsrc(data = mtcars)</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>rfsrc(data = mtcars)</td>
</tr>
<tr>
<td>Breiman (Shi-Horvath)</td>
<td>rfsrc(data = mtcars, method = &quot;sh&quot;)</td>
</tr>
</tbody>
</table>

### 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
mtry   nodesize
1      5
```

**tune.nodesize** Find the optimal nodesize

```
o$optimal.nodesize
```

### Grow

Convenient interface for growing a CART tree

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

#### Fast OpenMP parallel computing of random forests

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

### Prediction

#### Restore Mode

```
o <- rfsrc(Ozone ~ ., data = aqirquality)
> o$predpredicted
```

#### Predict on New Data

```
o$predPredict <- predict(object = o, newdata)
```

#### Prediction Type

- **Besides** `$predicted` and `$predicted.oob`, we also have `$class` and `$class.oob` for class labels in classification problems
- 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
- `$get.mv.predicted` returns predicted value for multivariate regression analysis

---

**clean up and impute data**

<table>
<thead>
<tr>
<th><code>$o$y</code></th>
<th><code>$o$z</code></th>
<th><code>$o$za</code></th>
<th><code>$o$za$y</code></th>
<th><code>$o$za$y$z</code></th>
<th><code>$na</code></th>
<th><code>$na$y</code></th>
<th><code>$na$za</code></th>
<th><code>$na$za$y</code></th>
<th><code>$na$za$y$z</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>o &lt;- rfsrc(y ~ a + z, data = dta, ntree = 1)</code></td>
<td><code>o &lt;- rfsrc(y ~ a + z, data = dta)</code></td>
<td><code>o &lt;- rfsrc(y ~ a + z, data = dta, na.action = &quot;na.impute&quot;)</code></td>
<td><code>o &lt;- rfsrc(y ~ a + z, data = dta)</code></td>
<td><code>o &lt;- rfsrc(y ~ a + z, data = dta)</code></td>
<td><code>NA</code></td>
<td><code>NA</code></td>
<td><code>NA</code></td>
<td><code>NA</code></td>
<td><code>NA</code></td>
</tr>
</tbody>
</table>
Variable Selection (VIMP)

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

- `o$serr.rate` returns tree cumulative OOB error rate; `print(o)` lists OOB error rate in the bottom; `plot(o)` plots OOB error rate along with number of trees
- `get.mv.error` obtains error rate from a multivariate random forest
- `o$importance` returns permutation VIMP and `plot(o)` plots VIMP when setting importance to "permute" or "TRUE" in `rfsrc` or `vimp`

Partial Plot

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

For categorical predictors:
```
plot.variable(o, xvar.names, partial = TRUE)
```

For continuous predictors:
```
plot.variable(o, xvar.names, partial = TRUE) and `partial`
```

Maximal Subtree Information

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

Variable Selection and Hunting

```
var.select(formula, data, method)
```

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

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

Prediction Error

```
o <- rfsrc(mpg ~ ., data = mtcars)
```

- `o$serr.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.mv.error` obtains error rate from a multivariate random forest

Inference from Trees

Split Statistics

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

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))`