A Short Introduction to the OmicsMarkeR Package

Charles Determan Jr.
cdetermanjr@hotmail.com

June 5, 2014

1 Introduction

The OmicsMarkeR package contains functions to streamline the analysis of 'omics' level datasets with the objective to classify groups and determine the most important features. OmicsMarkeR loads packages as needed and assumes that they are installed. I will provide a short tutorial using the both synthetic datasets created by internal functions as well as the 'Sonar' dataset. Install OmicMarkeR using

```
> # currently in development
> # install.packages("OmicsMarkeR", dependencies = c("Imports"))
```

to ensure that all the needed packages are installed.

You may also install the developmental version using

```
> library(devtools)
> install_github("OmicsMarkeR", username = "cdeterman")
```

2 Example

OmicsMarkeR has a few simplified functions that attempt to streamline the classification and feature selection process including the addition of stability metrics. We will first generate a synthetic dataset. This includes three functions that can be used to create multivariate datasets that can mimic specific omics examples. This can include a null dataset via `nvar = 50` and `nsamp = 100`.

The `create.corr.matrix` function induces correlations to the datasets. The `create.discr.matrix` function induces variables to be discriminate between groups. The number of groups can be specified with `num.groups`.

```
> library("OmicsMarkeR")
> set.seed(123)
> dat.discr <- create.discr.matrix(
+   create.corr.matrix(
+     create.random.matrix(nvar = 50,
+                          nsamp = 100,
+                          st.dev = 1,
+                          perturb = 0.2)),
+   D = 10
+   )$discr.mat
```

To avoid confusion in the coding, one can isolate the variables and classes from the newly created synthetic dataset. These two objects are then used in the `fs.stability` function. I can then choose which algorithm(s) to apply e.g. `method = c("plsda", "rf")`, the number of top important features `f = 20`, the number of bootstrap reptititions for stability analysis `k = 3`, the number of k-fold cross-validations `k.folds = 10` and if I would like to see the progress output `verbose = TRUE`.

```
> vars <- dat.discr[,1:(ncol(dat.discr)-1)]
> groups <- dat.discr[,ncol(dat.discr)]
> results <- fs.stability(vars,
+                         groups,
+                         method = c("plsda", "rf"),
+                         f = 10,
+                         k = 3,
+                         k.folds = 10,
+                         verbose = FALSE)

[1] "Model Tuning Complete"

Aggregating results
Selecting tuning parameters
[1] "plsda complete"

Aggregating results
Selecting tuning parameters
[1] "rf complete"

Aggregating results
Selecting tuning parameters
Calculating Model Performance Statistics
```

If I would like to see the performance metrics, I can simply use the `performance.metrics` function. This will provide a concise data.frame of confusion matrix and ROC statistics. Additionally, the Robustness-Performance Trade-off value is provided with the results `RPT`.

```
> performance.metrics(results)

Model Performance Statistics

                       plsda         rf
Accuracy           0.88333333 0.93333333
Kappa              0.76666667 0.86666667
ROC.AUC            0.92000000 0.96000000
Sensitivity        0.86666667 0.93333333
Specificity        0.90000000 0.93333333
Pos Pred Value     0.90303030 0.93333333
Neg Pred Value     0.87777778 0.93333333
Accuracy SD        0.07637626 0.11547005
Kappa SD           0.15275252 0.23094011
ROC.AUC SD         0.07549834 0.06928203
Sensitivity SD     0.11547005 0.11547005
Specificity SD     0.10000000 0.11547005
Pos Pred Value SD 0.10013765 0.11547005
Neg Pred Value SD 0.10715168 0.11547005

> results$RPT

    plsda        rf
0.7933472 0.4307692
```

3 Manual Grid

If the user would prefer to tune different models to different resolutions one can simply use the `denovo.grid` function to generate the level of resolution for each model, subsequently append the list together and submit as the `grid` parameter of `fs.stability`. For example, say the user would like to tune 15 components of the PLSDA model but only 3 levels of SVM.

```
> grid <- denovo.grid(vars, "plsda", 15)
> grid <- append(grid, denovo.grid(vars, "svm", 3))
> grid

$plsda
   .ncomp
1       1
2       2
3       3
4       4
5       5
6       6
7       7
8       8
```

```
9       9
10     10
11     11
12     12
13     13
14     14
15     15

$svm
    .C
1 0.25
2 0.50
3 1.00
```

```
> results <- fs.stability(vars,
+                         groups,
+                         grid = grid,
+                         method = c("plsda", "svm"),
+                         f = 10,
+                         k = 3,
+                         k.folds = 10,
+                         verbose = FALSE)

[1] "Model Tuning Complete"

Aggregating results
Selecting tuning parameters
[1] "plsda complete"

Aggregating results
Selecting tuning parameters
[1] "svm complete"

Aggregating results
Selecting tuning parameters
Calculating Model Performance Statistics
```

If the user would prefer to set a particular range for the parameters, such as components 3-8 in the PLSDA model, one can simply use the `expand.grid` function for which `denovo.grid` is simply a wrapper function. `denovo.grid`, however, has been designed to create grids with acceptable increments and relationships between parameters, with particular note to Random Forest, Gradient Boosting Machines, and Support Vector Machines and is generally recommended.

```
> grid <- list(expand.grid(.ncomp = seq(3,8)))
> grid <- append(grid, list(expand.grid(.C = c(.50, 1))))
> names(grid) <- c("plsda", "svm")
```

NOTE!!! The user should make sure to add the '.' before the parameter as this is important for internal consistency of the functions. Furthermore, the user currently must specify all optimizable

parameters (package will possibly implement defaults in the future for this scenario). This method can be used to set any combination of parameters desired given the user is familiar with choosing and optimizing such parameters. The parameters are the same as the native packages and can be easily accessed with the `params` function.