Benchmarking Binary Classifiers Algorithms

José Tamez-Peña

jose.tamezpena@tec.mx
Tecnológico de Monterrey, Escuela de Medicina,
Av. Morones Prieto No. 3000, Colonia Los Doctores CP 64710
Monterrey Nuevo León, México.

Table of Contents

Benchmarking with FRESA.CAD

This vignette shows the use of FRESA.CAD to evaluate the performance of several binary classification algorithms on a user-supplied data set.

The `FRESA.CAD::BinaryBenchmark(…)` function evaluates the holdout cross-validation[1] (CV) performance of the following algorithms:

- **B**ootstrap **S**tage-**Wi**se **M**odel **S**election (BSWiMS), or user-supplied cross-validated (CV) method.
- **L**east **A**bsolute **S**hrinkage and **S**election **O**perator[2] (LASSO)[1],
- **R**andom **F**orest[3] (RF)[2],
- **R**ecursive **P**artitioning and **R**egression **T**rees[4] (RPART)[3],
- **K N**earest **N**eighbors (KNN)[4] with BSWiMS features,
- **S**upport **V**ector **M**achine[5] (SVM)[5] with **m**inimum-**R**edundancy-**M**aximum-**R**elevance[6] (mRMR)[6] feature selection filter,
- The ensemble of all the above methods.

Furthermore, the function evaluates the effect of the following feature selection algorithms: BSWiMS, LASSO, RPART, RF, integrated discrimination improvement[7] (IDI), net reclassification improvement (NRI), t student test, Wilcoxon test, Kendall correlation, and mRMR as filters on the following classifiers: KNN, naive Bayes[7], nearest centroid (NC) with normalized root sum square distance and Spearman correlation distance, RF and SVM. Appendix A shows the workflow of the Benchmark procedure. The default parameters of the `BinaryBenchmark(…)` uses BSWiMS, but the user has the freedom to change the BSWiMS method for any data classifier with a predict method as show in Appendix B. Internally `BinaryBenchmark(…)` runs the `FRESA.CAD::randomCV(…)` on every filter/classifier using the same train/test split. The test results of the CV are accumulated and evaluated by the `FRESA.CAD::predictionStats_binary(…)` function. The evaluation

[1] LASSO algorithm uses the **glmnet** package. The `glmnet::cv.glmnet(…)` function is run to fit the LASSO algorithm with default parameters and predictions are done using min se.
[2] RF uses the **randomForest** package. The `randomForest::randomForest(…)` function with default parameters is run to fit the data.
[3] RPART uses the **rpart** package. The `rpart::rpart(…)` function is used to fit the train data with default parameters.
[4] KNN uses the **class** package. `FRESA.CAD::KNN_method(…)` prepares the trained data with order statistics for data normalization and set k to the square root of number of subjects. Predictions are done using `class::knn(…)` function of the **class** package.
[5] SVM uses the e1071 package. The `e1071::svm(…)` fit function is used with default parameters.
[6] mRMR uses the RMRe package. `FRESA.CAD::mRMR.classic_FRESA(…)`function wraps the classic mRMR algorithm `mRMRe::mRMR.classic(…)`setting the maximum number of features to return.
[7] Naïve Bayes algorithm uses the **naivebayes** package. `FRESA.CAD::NAIVE_BAYES(…)`function wraps the `naivebayes::naive_bayes(…)` function and fit the data with default parameters.

function returns the accuracy, precision, sensitivity, the balanced error rate and the ROC area under the curve with their corresponding 95% confidence intervals (95%CI)

The simplest way to visualize the results of the `BinaryBenchmark(…)` function is using the provided `plot(…)` function. The plot function compares the statistics and ranks them by comparing the 95%CI, as well as returns the ranked tables associated to each metric. Each method accumulates a positive score each time their lower CI is superior to the mean of the other methods and loses a point each time the mean is inferior to the top 95%CI of the other methods. The ranked data is visualized by bar plots that include the 95%CI.

The simplest code for data evaluation is the following:

```
R> cp <- BinaryBenchmark(theData = Data, theOutcome = "Class")
R> pr <-plot(cp)
```

The above code will randomly split the data into train and test sets 100 times. Then the results of the 100 test evaluation will be ranked and plotted by the `plot.FRESA_benchmark(x,…)` function. The user has the freedom to select the number of random train/test splits as well as the fraction of the data that will be used at each train session. The runtime will depend on the size of the data sets. It may take a few minutes for small data sets to hours for larger sets.

The next section will use the colon cancer data set to show the usage of the method and to display the results.

Sample run: The Colon Cancer Data Set

The data is an instance for the Alon et al. (1999) colon cancer data. 62 samples (40 tumor samples, 22 normal samples) from colon-cancer patients were analyzed with an Affymetrix oligonucleotide Hum6000 array[8]. Here we will use the data to show the ability of the benchmarking/plot functions to evaluate the performance of the classifier algorithms.

FRESA.CAD requires that all the data sets to be evaluated be R data frames, where each column is a feature and each row is a sample. The data frame has to be numeric, hence each factor has to be converted to its corresponding numeric value. The "rda" R package has the Colon dataset:

```
R> data(colon,package = "rda")
R> Colon <- as.data.frame(cbind(Class = colon.y,colon.x))
R> Colon$Class <- Colon$Class - 1
```

After loading the data, I make a numeric data frame where cases are 1s and controls are the 0s. The next step returned the number of informative features using the `FRESA.CAD::univariate_Logit(…)` function.

```
R> #Univariate feature selection snippet

R> q_values <- univariate_Logit(data = Colon,
                                Outcome = "Class",
                                pvalue = 0.05)

R> pander::pander(length(q_values))
```

The function returned 495 features that have the potential to separate cases and controls. The histogram of the p-values is show next:

```
R> hist(-log10(q_values),main = "Distribution of q-values")
```

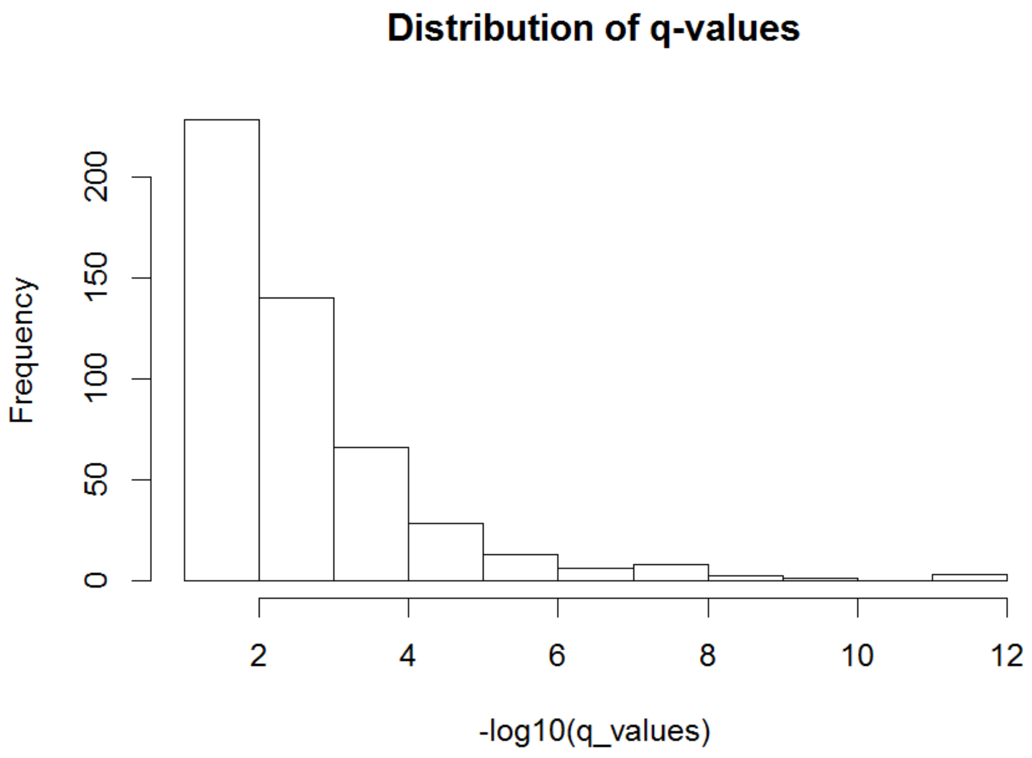

Figure 1: Distribution of the significant q-values (q<0.05) in the colon data set.

The Benchmarking Function

Once I have prepared the data frame, I can compare the classification algorithms using the benchmark function:

```
R> #Benchmarking the Colon Dataset

R> cp <- BinaryBenchmark(theData = Colon, theOutcome = "Class", reps = 75,
trainFraction = 0.8)
```

The above example specifies that 80% of the data will be used for training and the CV will be repeated 75 times. The above configuration will result, on average, 15 test results per sample. Internally the CV function will report the median of the repeated test and provide a single test result for each subject. Figure 2 shows the plots that the benchmark function outputs while running. The plots are ROC curves of the ensemble of the 75 test results. The ensemble provides a unique test prediction of each subject, hence it is possible to estimate a reliable 95%CI of each metric.

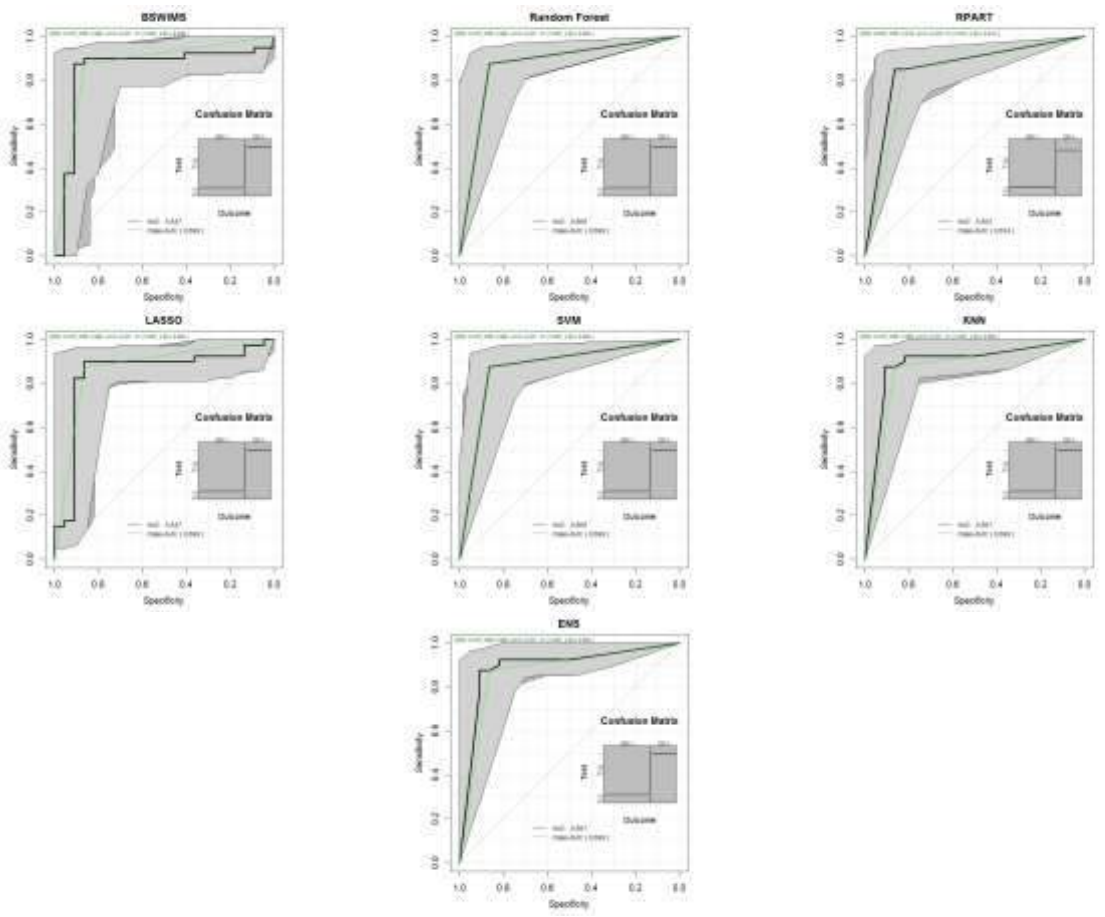

Figure 2: Plot outputs of the `FRESA.CAD::BinaryBenchmark(…)` showing the ROC of the main classifiers

Analyzing the Results

The results of the `BinaryBenchmark(…)`object can be visualized in different forms. This section describes the heat-map and the plot FRESA.CAD functions. But first, here is the average train time of each classifier:

```
R> #The Runtimes
R> pander::pander(cp$cpuElapsedTimes)
```

BSWiMS	RF	RPART	LASSO	SVM	KNN	ENS
8.834	2.044	0.4968	0.2955	0.01187	0.0268	11.71

As we can see the BSWiMS method took on average 8.3 seconds to train. The fastest method was the KNN that only took 0.014 seconds to train on 80% of the data.

Regarding the similarity of the test results, I used the `FRESA.CAD::heatMaps(…)` to display the similarity of the test results. Figure 3 shows each one of the test predictions by the different classification methods. It is clear that all the methods had similar predictions on each subject, but the RPART method. The RPART had slightly different predictions of a selected group of subjects.

```
R> # Heat maps of predicted values

R> hm <- heatMaps(Outcome = "Outcome",
              data = cp$testPredictions,
              title = "Heat Map",Scale = TRUE,
              hCluster = "col",
              cexRow = 0.25,cexCol = 0.75,srtCol = 45)
```

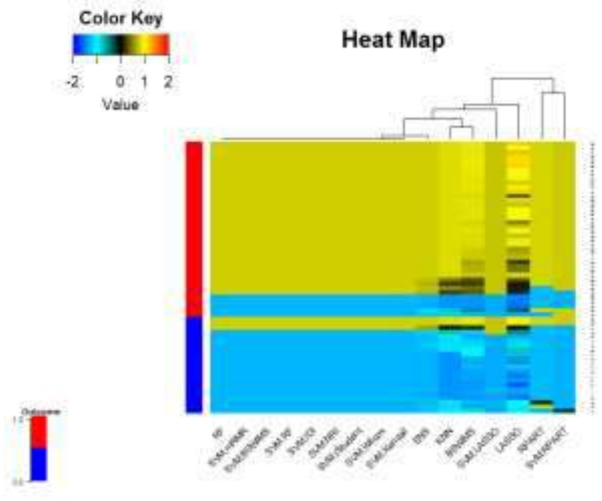

Figure 3: Heat map showing how similar is each classifier.

The object returned by the benchmarking can be visualized by invoking the `FRESA.CAD::plot(x,…)` function. This function will output a series of plots that visualize the key metrics of test results. Figure 4 to 6 shows the outputs returned for Feature Selection evaluation (Figure 4), bar plots for the basic classifiers (Figure 5) and the bar plots evaluating the Filter/Classifier performance (Figure 6).

```
R> pr <- plot(cp)
```

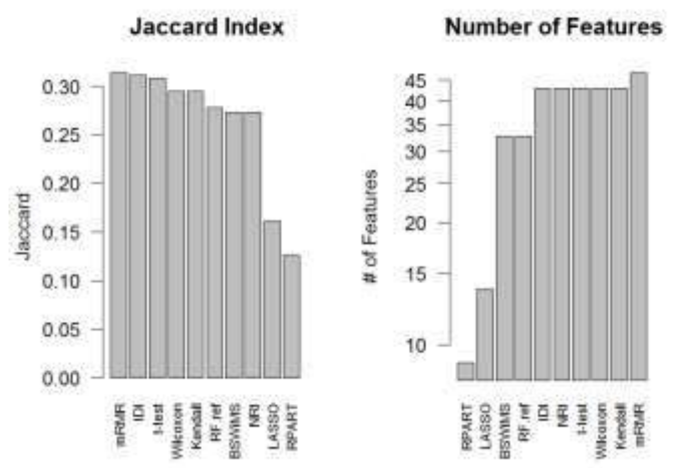

Figure 4: The Jaccard index and the number of returned features of each feature selection algorithm.

Figure 4 shows the Jaccard index of the different feature selection algorithms tested by the FRESA.CAD benchmarking function. This index indicates how consistent the selected features on all the CV runs were. The right plot shows the number of selected features by the `FRESA.CAD::filterUnivariate` functions. These FRESA.CAD functions are run inside the benchmark with an upper limit to the returned number of features. For that reason, no more than 45 features were returned by the feature selection methods.

Figure 5 shows the output of the performance comparison of the stand-alone classifiers. The analysis of the 95%CI indicates that none of the methods were superior or inferior to any other method. While Figure 6 shows the comparison of all the filtered/classifiers. These last plots show that there are filters/classifiers method inferior to the others.

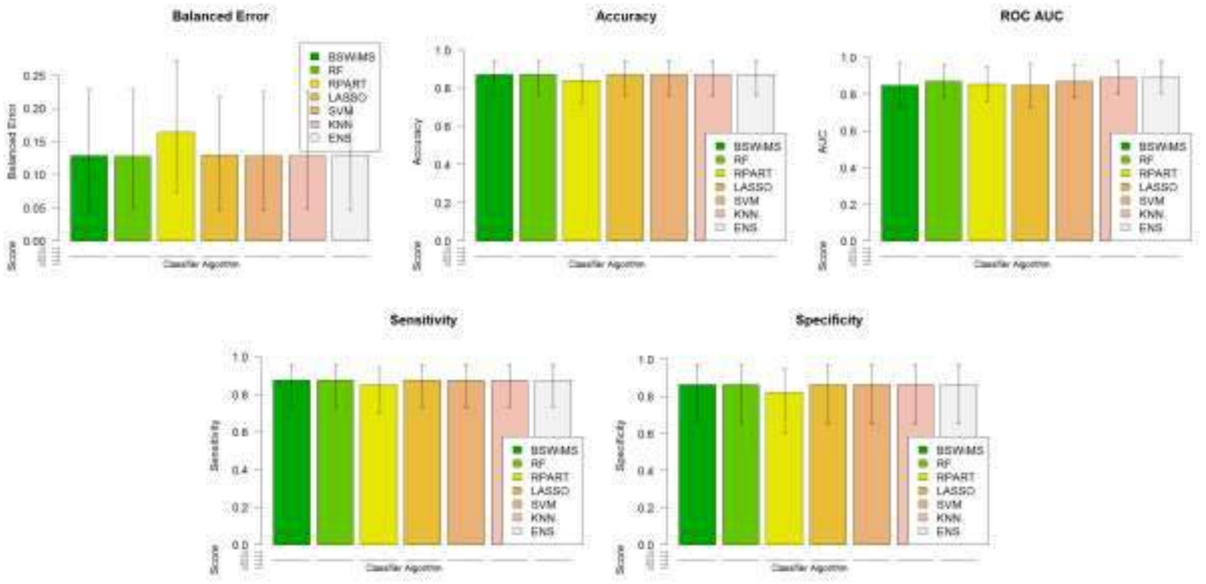

Figure 5: Balanced error rate, Accuracy, ACU, Sensitivity and Specify of the analyzed classifiers.

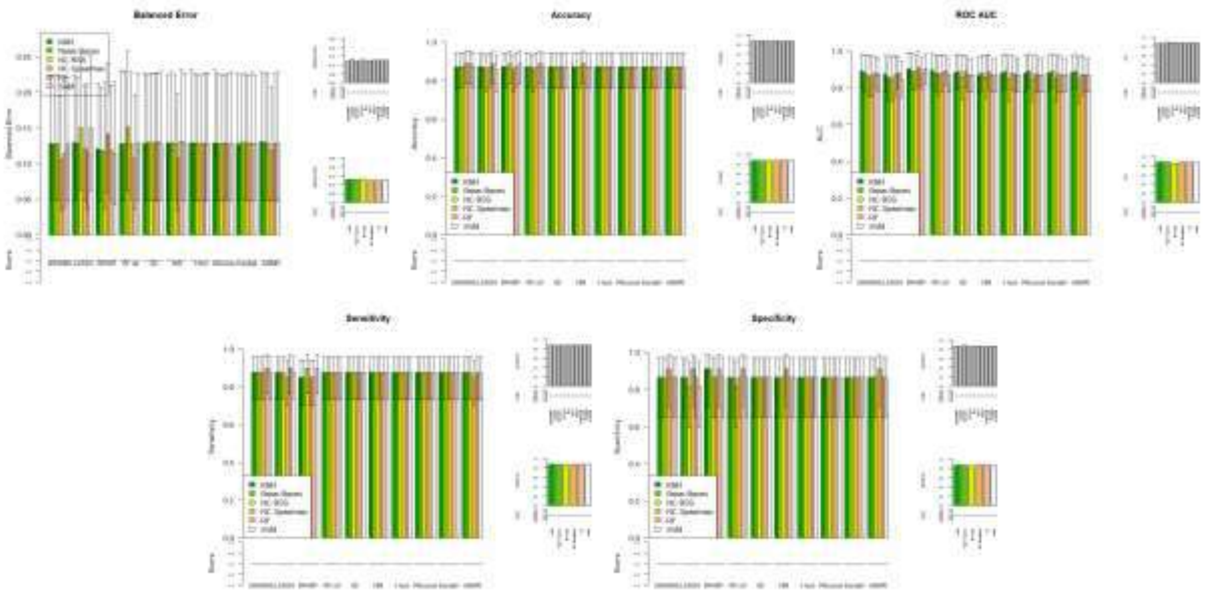

Figure 6: Balanced Error Rate, Accuracy, AUC, Sensitivity, and Specificity for the different combinations of Filters/Classifiers algorithms.

Meta-analysis: Radar Plots for Summarizing Performance Results

All the presented results can be summarized by using radar plots as shown in Figure 7. These two plots show that the performance the methods are very similar. The largest difference in CPU time, and the number/consistency of the selected features. The R code snippet used to create radar plots is shown in Appendix C.

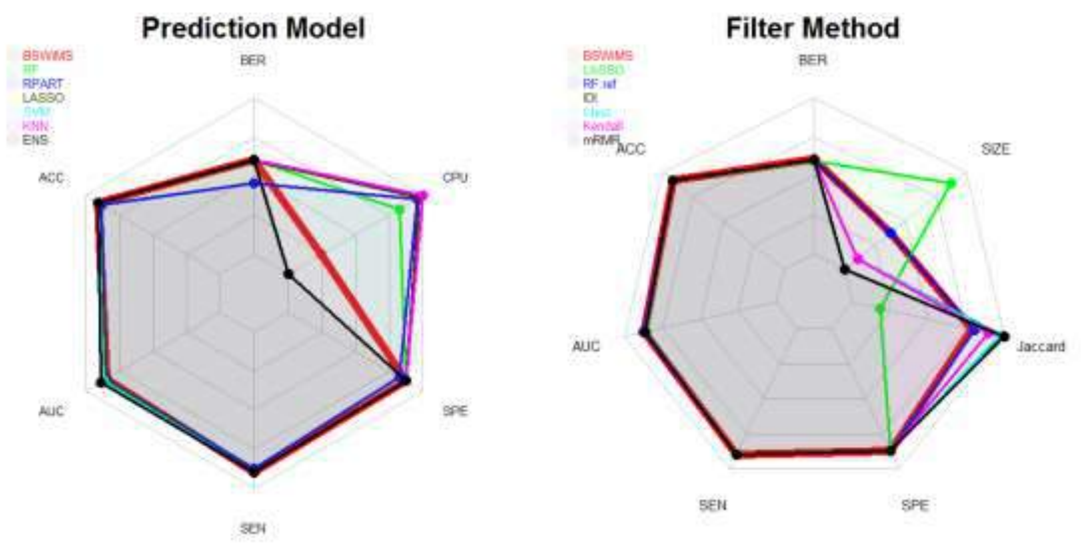

Figure 7: Radar plots summarizing the results of the benchmark analysis.

Meta-analysis: Which are the Relevant Features?

The returned object of the Benchmark function contains a table with all the selected features by the method. I will use the `gplots::heatmap.2(…)` function to display the features that were selected more than 10%. Figure 8 shows the result of the analysis. Figure 9 shows the heat map of the features and their association with the outcome. Finally, Table 1 shows the descriptive statistics of the top 10 features as well as the average selection rate of these features.

```
R> rm <- rowMeans(cp$featureSelectionFrequency)
R> selFrequency <- cp$featureSelectionFrequency[rm > 0.1,]
R> gplots::heatmap.2(selFrequency,trace = "none",
                  mar = c(10,10),
                  main = "Features",cexRow = 0.25)
```

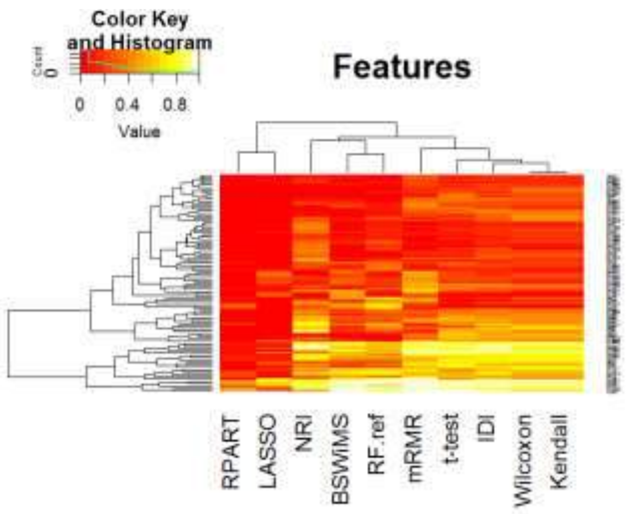

Figure 8: The features with a selection rate greater than 10%.

```
R> hm <- heatMaps(Outcome = "Class",
               data = Colon[,c("Class",rownames(selFrequency))],
               title = "Heat Map",Scale = TRUE,
               hCluster = "col",cexRow = 0.25,cexCol = 0.25,srtCol = 45)
```

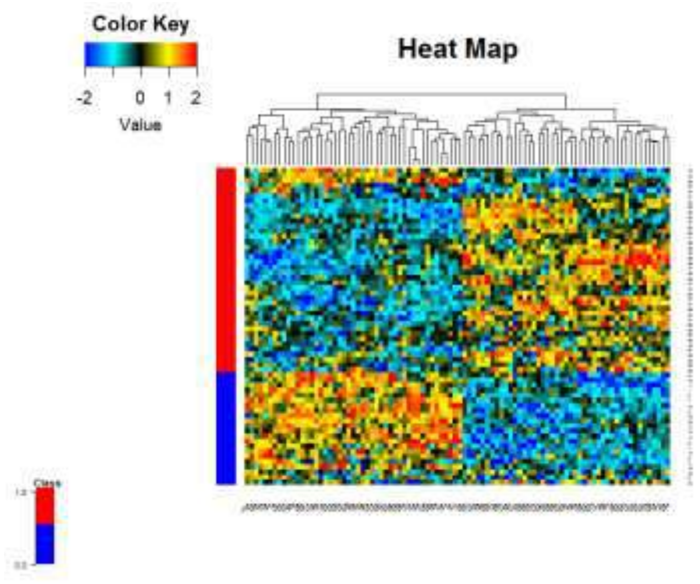

Figure 9: `FRESA.CAD::heatMaps(…)` showing the value of the top features discovered by FS methods.

Table 1: Top 10 Features

	controlMean	controlStd	caseMean	caseStd	ROCAUC	Wilcon pvalue	Frequency
V378	0.741	0.4201	-0.0097	0.3183	0.9341	0	0.8853
V494	1.207	0.5516	0.0745	0.516	0.9205	0	0.8667
V1636	-0.07	0.4481	-1.098	0.5804	0.9045	0	0.804
V1424	1.121	0.8332	-0.3798	0.8767	0.8909	0	0.6253
V626	0.2895	0.5038	1.138	0.482	0.8898	0	0.6853
V250	2.091	0.7986	0.7251	0.637	0.8886	0	0.7347
V1772	-1.061	0.4778	-0.2618	0.5116	0.8852	0	0.7053
V1043	-0.5278	0.3595	0.2742	0.6335	0.8795	0	0.5133
V1844	0.3261	0.5442	-0.8701	0.8825	0.8761	0	0.56
V1773	-1.319	0.45	-0.6071	0.4575	0.8727	0	0.7933

A snippet of the code used to generate Table 1:

```
R> #Setting up a summary table with relevant information

R> vlist <- rownames(selFrequency)
R> vlist <- cbind(vlist,vlist)
R> univ <- univariateRankVariables(variableList = vlist,
                                   formula = "Class~1",
                                   Outcome = "Class",
                                   data = Colon,
                                   type = "LOGIT",
                                   rankingTest = "zIDI",
                                   uniType = "Binary")

R> univ <- univ[,c("controlMean","controlStd","caseMean","caseStd","ROCAUC","WilcoxRes.p")]

R> cnames <- colnames(univ);
R> univ <- cbind(univ,rm[rownames(univ)])
R> colnames(univ) <- c(cnames,"Frequency")
R> univ <- univ[order(-univ[,5]),]
R> pander::pander(univ[1:10,],caption = "Features",round = 4)
```

Conclusion

FRSA.CAD provides a simple to use set of functions that evaluate, and display the performance of supervised classifiers on a user-provided data set. The results of the call and the provided visual results gives the user a perspective of what is the best classifier, the expected performance on an independent test and the set of features that are associated with the study outcome.

References

1. Kim JH. Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. Computational Statistics & Data Analysis. 2009;53(11):3735-45. doi: 10.1016/j.csda.2009.04.009. PubMed PMID: WOS:000267505600001.
2. Tibshirani R. Regression shrinkage and selection via the Lasso. Journal of the Royal Statistical Society Series B-Methodological. 1996;58(1):267-88. PubMed PMID: WOS:A1996TU31400017.
3. Breiman L. Random forests. Machine Learning. 2001;45(1):5-32. doi: 10.1023/a:1010933404324. PubMed PMID: WOS:000170489900001.
4. De'ath G, Fabricius KE. Classification and regression trees: A powerful yet simple technique for ecological data analysis. Ecology. 2000;81(11):3178-92. doi: 10.1890/0012-9658(2000)081[3178:cartap]2.0.co;2. PubMed PMID: WOS:000165384000018.
5. Cortes C, Vapnik V. SUPPORT-VECTOR NETWORKS. Machine Learning. 1995;20(3):273-97. doi: 10.1007/bf00994018. PubMed PMID: WOS:A1995RX35400003.
6. Ding C, Peng H. Minimum redundancy feature selection from microarray gene expression data. Journal of bioinformatics and computational biology. 2005;3(2):185-205. doi: 10.1142/s0219720005001004. PubMed PMID: MEDLINE:15852500.
7. Pencina MJ, D'Agostino RB, Sr., D'Agostino RB, Jr., Vasan RS. Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond. Statistics in Medicine. 2008;27(2):157-72. doi: 10.1002/sim.2929. PubMed PMID: WOS:000253098900001.
8. Alon U, Barkai N, Notterman DA, Gish K, Ybarra S, Mack D, et al. Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. Proceedings of the National Academy of Sciences of the United States of America. 1999;96(12):6745-50. doi: 10.1073/pnas.96.12.6745. PubMed PMID: WOS:000080842200032.

Appendix A: The Benchmark Workflow

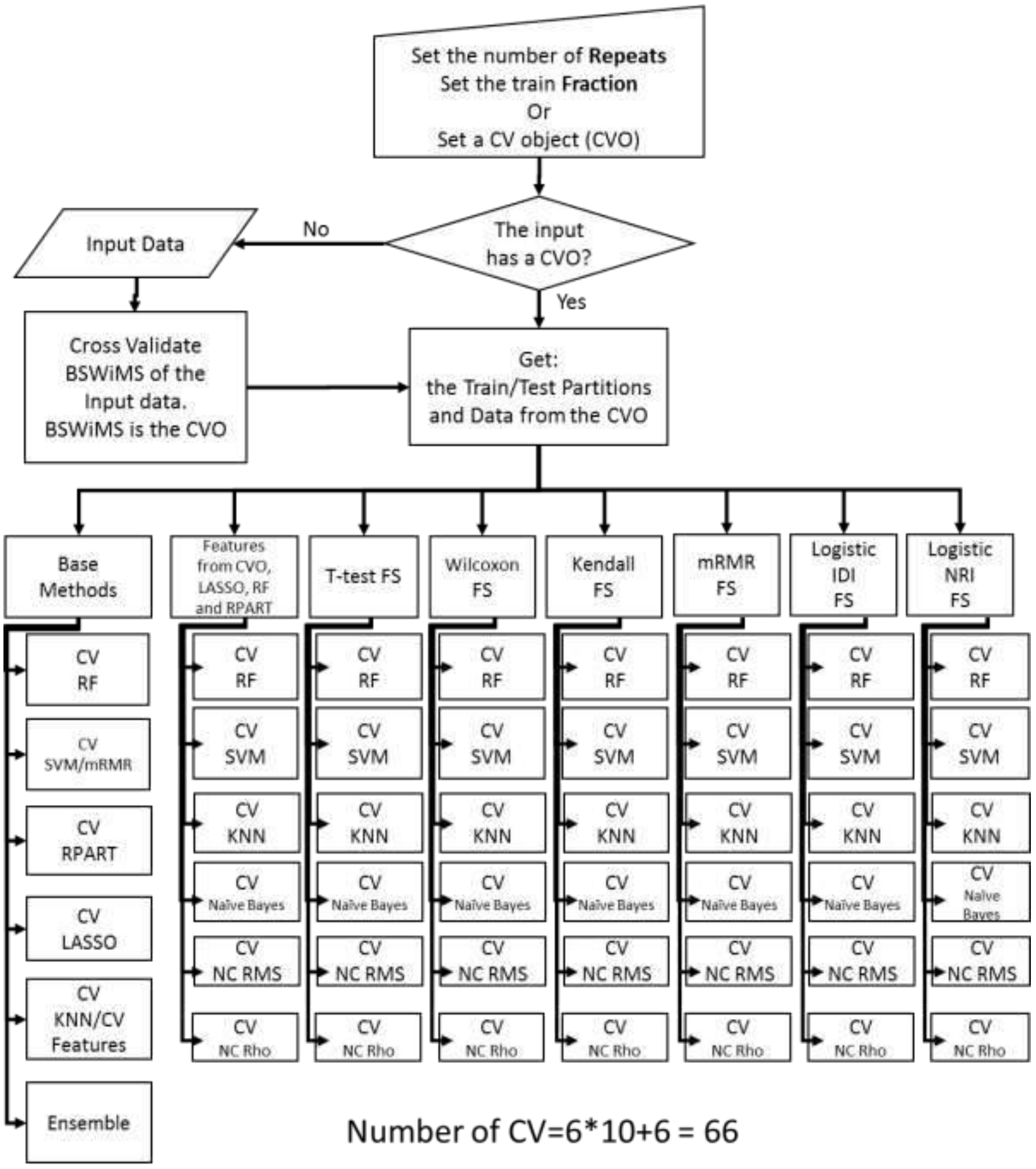

Figure 10: Holdout Cross Validations performed by the Binary Benchmark Function

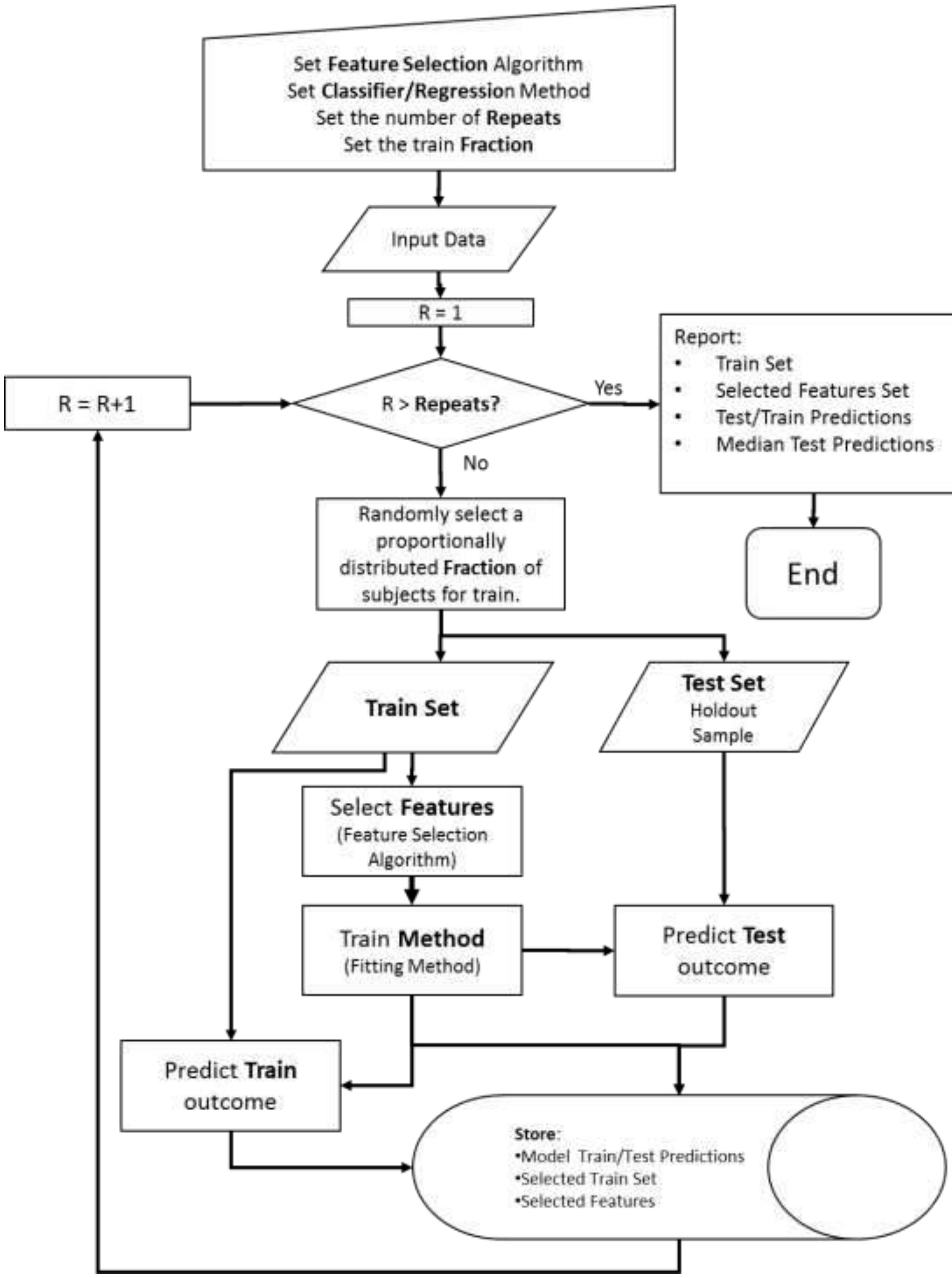
Set Feature Selection Algorithm
Set Classifier/Regression Method
Set the number of Repeats
Set the train Fraction
Input Data
R = 1
R = R+1
R > Repeats?
Yes
No
Report:
• Train Set
• Selected Features Set
• Test/Train Predictions
• Median Test Predictions
End
Randomly select a proportionally distributed Fraction of subjects for train.
Train Set
Test Set Holdout Sample
Select Features (Feature Selection Algorithm)
Train Method (Fitting Method)
Predict Test outcome
Predict Train outcome
Store:
•Model Train/Test Predictions
•Selected Train Set
•Selected Features

Figure 11: Random Holdout Cross Validation

Appendix B: Benchmarking a User-Specified Classifier

The `BinaryBenchmark(…)` function can be set up to evaluate any supervised classifier as long as the classifier has the proper R `x<-fit(data, formula,…)` method with its corresponding `predict.fit(x,…)` function. This section shows how to change the base classifier to a quadratic discriminant analysis (QDA) using the `MASS::qda(…)` fit function.

The basic instructions are show in the following snippet:

```
R> cv <- randomCV(Colon,"Class",
               MASS::qda,trainFraction = 0.8,
               repetitions = 75,
               featureSelectionFunction = univariate_Wilcoxon,
               featureSelection.control = list(limit = 0.25,thr = 0.95));

R> ps <- predictionStats_binary(cv$medianTest,plotname = "QDA",cex=0.8)


R> cp <- BinaryBenchmark(referenceCV = cv,referenceName =
"QDA",referenceFilterName="Wilcoxon_25")

R> pr <- plot(cp)
```

The above snippet shows how to cross-validate and benchmark the QDA classifier on the Colon dataset. First, the QDA CV was done using 80% of the data for training and 20% for testing and the Wilcoxon test was used as a feature selection method. The feature selection was constrained to return as many of 25% of the number of samples in the data set, by step-wise removing highly correlated features (Spearman rho > 0.95^n, where n is the step). This constraint is required by the QDA method in order to be able to compute the class covariance matrix. The CV was repeated 75 times. Hence, on average each subject had 15 test results. Second, the `predictionStats_binary(…)` function was used to create Figure 10. This figure shows the ROC plot of CV test results, as well as the confusion matrix. The same function returns the accuracy, sensitivity, specificity, ROC AUC and the balanced error rate with their corresponding 95%CI. Third, the `BinaryBenchmark(…)` function is run passing the CV object and the names associated with the classification method and the feature selection method. The function call will use the QDA CV train/test splits that were repeated 75 times in all the classifiers. Finally, the `plot(cp)` call returns the similar set of plots shown in Figures 5 and 6 but specific to the QDA process. Figure 11 shows the specific plot associated with the balanced error rate.

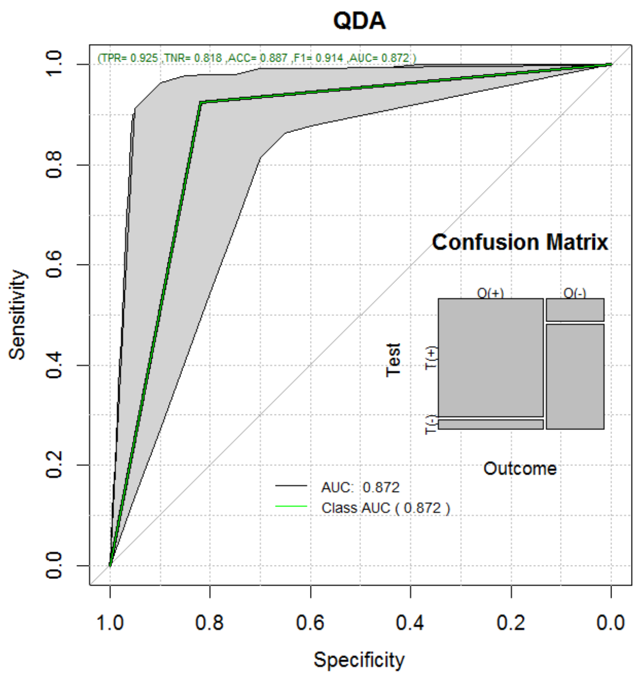

Figure 12: ROC plot of the QDA on the Cancer data set.

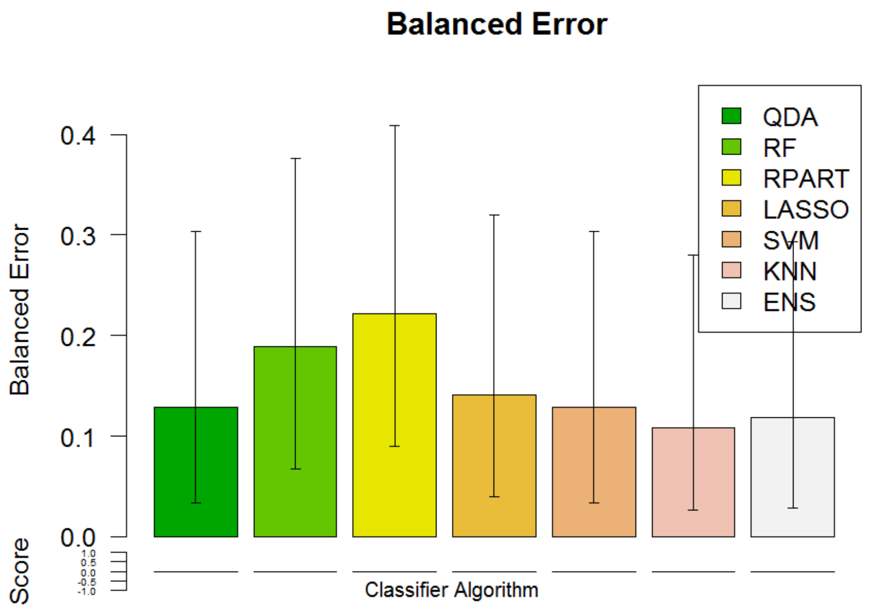

Figure 13: Balanced Error Rate of the QDA, RF, RPART, LASSO, SVM, KNN and the ensemble on the Cancer dataset.

Appendix C: The Radar Plots Code

```
R> op <- par(no.readonly = TRUE)

R> par(mfrow = c(1,2),xpd = TRUE,pty = "s",mar = c(1,1,1,1))

R> mNames <- names(cp$cpuElapsedTimes)

R> classRanks <- c(pr$minMaxMetrics$BER[1],
                 pr$minMaxMetrics$ACC[2],
                 pr$minMaxMetrics$AUC[2],
                 pr$minMaxMetrics$SEN[2],
                 pr$minMaxMetrics$SPE[2],
                 min(cp$cpuElapsedTimes))

R> classRanks <- rbind(classRanks,c(pr$minMaxMetrics$BER[2],0,0,0,0,max(cp$cpuElapsedTimes)))
R> classRanks <- as.data.frame(rbind(classRanks,
                                 cbind(t(pr$metrics[c("BER","ACC","AUC","SEN","SPE"),mNames]),
                                       cp$cpuElapsedTimes)))

R> colnames(classRanks) <- c("BER","ACC","AUC","SEN","SPE","CPU")

R> classRanks$BER <- -classRanks$BER
R> classRanks$CPU <- -classRanks$CPU

R> colors_border = c( rgb(1.0,0.0,0.0,1.0),
                    rgb(0.0,1.0,0.0,1.0),
                    rgb(0.0,0.0,1.0,1.0),
                    rgb(0.2,0.2,0.0,1.0),
                    rgb(0.0,1.0,1.0,1.0),
                    rgb(1.0,0.0,1.0,1.0),
                    rgb(0.0,0.0,0.0,1.0) )

R> colors_in = c( rgb(1.0,0.0,0.0,0.05),
                rgb(0.0,1.0,0.0,0.05),
                rgb(0.0,0.0,1.0,0.05),
                rgb(1.0,1.0,0.0,0.05),
                rgb(0.0,1.0,1.0,0.05),
                rgb(1.0,0.0,1.0,0.05),
                rgb(0.0,0.0,0.0,0.05) )

R> fmsb::radarchart(classRanks,axistype = 0,
           maxmin = T,pcol = colors_border,
           pfcol = colors_in,plwd = c(6,2,2,2,2,2,2),
           plty = 1, cglcol = "grey", cglty = 1,
           axislabcol = "black",cglwd = 0.8, vlcex  = 0.5,
           title = "Prediction Model")

R> legend("topleft",legend = rownames(classRanks[-c(1,2),]),
       bty = "n",pch = 20,col = colors_in,
       text.col = colors_border,cex = 0.5,pt.cex = 2)


R> filnames <- c("BSWiMS","LASSO","RF.ref","IDI","t-test","Kendall","mRMR")

R> filterRanks <- c(pr$minMaxMetrics$BER[1],
                  pr$minMaxMetrics$ACC[2],
                  pr$minMaxMetrics$AUC[2],
                  pr$minMaxMetrics$SEN[2],
                  pr$minMaxMetrics$SPE[2],
                  max(cp$jaccard),min(cp$featsize));

R> filterRanks <- rbind(filterRanks,
                      c(pr$minMaxMetrics$BER[2],0,0,0,0,min(cp$jaccard),max(cp$featsize)));

R> filterRanks <- as.data.frame(rbind(filterRanks,
                                  cbind(t(pr$metrics_filter[c("BER","ACC","AUC","SEN","SPE"),filnames]),
                                        cp$jaccard[filnames],cp$featsize[filnames])));
```

```
R> colnames(filterRanks) <- c("BER","ACC","AUC","SEN","SPE","Jaccard","SIZE")
R> filterRanks$BER <- -filterRanks$BER
R> filterRanks$SIZE <- -filterRanks$SIZE

R> colors_border = c(rgb(1.0,0.0,0.0,1.0),
                  rgb(0.0,1.0,0.0,1.0),
                  rgb(0.0,0.0,1.0,1.0),
                  rgb(0.2,0.2,0.0,1.0),
                  rgb(0.0,1.0,1.0,1.0),
                  rgb(1.0,0.0,1.0,1.0),
                  rgb(0.0,0.0,0.0,1.0) )

R> colors_in = c( rgb(1.0,0.0,0.0,0.05),
               rgb(0.0,1.0,0.0,0.05),
               rgb(0.0,0.0,1.0,0.05),
               rgb(1.0,1.0,0.0,0.05),
               rgb(0.0,1.0,1.0,0.05),
               rgb(1.0,0.0,1.0,0.05),
               rgb(0.0,0.0,0.0,0.05) )

R> fmsb::radarchart(filterRanks,axistype = 0,
           maxmin = T,pcol = colors_border,
           pfcol = colors_in,plwd = c(6,2,2,2,2,2,2),
           plty = 1, cglcol = "grey", cglty = 1,
           axislabcol = "black",cglwd = 0.8, vlcex  = 0.6,
           title = "Filter Method" )


R> legend("topleft",legend = rownames(filterRanks[-c(1,2),]),
       bty = "n",pch = 20,col = colors_in,
       text.col = colors_border,cex = 0.5,pt.cex = 2)

R> par(mfrow = c(1,1))
R> par(op)
```