Training and testing a K-Top-Scoring-Pair (KTSP) classifier with switchBox.

Bahman Afsari, Luigi Marchionni and Wikum Dinalankara

The Sidney Kimmel Comprehensive Cancer Center,
Johns Hopkins University School of Medicine

Modified: June 20, 2014. Compiled: November 24, 2026

Contents

1 Introduction 2

2 Installing the package 4

3 Data structure 4

3.1 Training set 4

3.2 Testing set 5

4 Training KTSP algorithm 6

4.1 Unrestricted KTSP classifiers 6

4.1.1 Default statistical filtering 6

4.1.2 Altenative filtering methods 9

4.2 Training a Restricted KTSP algorithm 11

5 Calculate and aggregates the TSP votes 13

6 Classifiy samples and compute the classifier performance 16

6.1 Classifiy training samples 16
6.2 Classifiy validation samples 18

7 Compute the signed TSP scores **19**

8 Use of deprecated functions **22**

9 System Information **25**

10 Literature Cited **26**

1 Introduction

The `switchBox` package allows to train and validate a K-Top-Scoring-Pair (KTSP) classifier, as used by Marchionni et al in [1]. KTSP is an extension of the TSP classifier described by Geman and colleagues [2, 3, 4]. The TSP algorithm is a simple binary classifier based on the ordering of two measurements.Basing the prediction solely on the ordering of a small number of features (e.g. gene expressions), known as ranked based methodology, seems a promising approach to to build robust classifiers to data normalization and rise to more transparent decision rules. The first and simplest of such methodologies, the Top-Scoring Pair (*TSP*) classifier, was introduced in [2] and is based on reversal of two features (e.g. the expressions of two genes). Multiple extensions were proposed afterwards, e.g. [3] and many of these extensions have been successfully applied for diagnosis and prognosis of cancer such as recurrence of breast cancer in [1]. A popular successor of *TSP* classifiers is *kTSP* ([3]), which applies the majority voting among multiple of the the reversal of pairs of features. In addition to being applied by peer scientists, *kTSP* shown its power by wining the ICMLA the challenge for cancer classification in the presence of other competitive methods such as Support Vector Machines ([5]).

kTSP decision is based on k feature (e.g. gene) pairs, say, $\Theta = \{(i_1, j_1), \ldots, (i_k, j_k)\}$. If we denote the feature profile with $\underline{X} = (X_1, X_2, \ldots)$, the family of rank based classifiers is an aggregation of the comparisons $X_{i_l} < X_{j_l}$. Specifically, the kTSP statistics can be written as:

$$\kappa = \{\sum_{l=1}^{k} I(X_{i_l} < X_{j_l})\} - \frac{k}{2},$$

where I is the indicator function. The kTSP classification decision can be produced by thresholding the κ, i.e. $\hat{Y} = I\{\kappa > \tau\}$ provided the labels $Y \in \{0, 1\}$. The standard threshold is $\tau = 0$. The only parameters required for calculating κ is the feature pairs. Usually, disjoint feature pairs are desirable because an outlier feature value cannot heavily influence the decision. In the introductory paper to kTSP ([**?**]), the authors proposed an ad-hoc method for feature selection. This method was based on score for each pair of features which measures how discriminative is a comparison of the feature values. If we denote the score related to the gene i and j by s_{ij}, then the score was defined as

$$s_{ij} = |P(X_i < X_j|Y = 1) - P(X_i < X_j|Y = 0)|.$$

We can sort the pairs of genes by this score. A pair with large score (close to one) indicates that the reversal of the feature value predicts the phenotype accurately.

In [6], an analysis of variance was proposed for gene selection in kTSP and other rank-based classifiers. This method finds the feature pairs which make the distribution of κ under two classes *far apart* in the analysis of variance sense. In mathematical words, we seek the set of feature pairs, Θ^*, that

$$\Theta^* = \arg\max_{\Theta} \frac{E(\kappa(\Theta)|Y = 1) - E(\kappa(\Theta)|Y = 0)}{\sqrt{Var(\kappa(\Theta)|Y = 1) + Var(\kappa(\Theta)|Y = 0)}}.$$

This method automatically chooses the number of genes and hence, it is almost a parameter free method. However, the search for Θ is very intensive search. So, a greedy and approximate search was proposed to find the optimal set of gene pairs. In practice, the only parameter required is a maximum cap for the number pairs, k.

The `switchBox` package contains several utilities enabling to:

1. Filter the features to be used to develop the classifier (*i.e.*, differentially expressed genes);
2. Compute the scores for all available feature pairs to identify the top performing TSPs;
3. Compute the scores for selected feature pairs to identify the top performing TSPs;
4. Identify the number of top pairs, K, to be used in the final classifier;

5. Compute individual TSP votes for one class or the other and aggregate the votes based on various methods;

6. Classify new samples based on the top KTSP based on various methods;

2 Installing the package

Download and install the package `switchBox` from Bioconductor.

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
      install.packages("BiocManager")
> BiocManager::install("switchBox")
```

Load the library.

```
> require(switchBox)
```

3 Data structure

3.1 Training set

Load the example training data contained in the `switchBox` package.

```
> ### Load the example data for the TRAINING set
> data(trainingData)
```

The object `matTraining` is a numeric matrix containing gene expression data for the 78 breast cancer patients and the 70 genes used to implement the MammaPrint assay [7]. This data was obtained from from the `MammaPrintData` package, as described in [1]. Samples are stored by column and genes by row. Gene annotation is stored as `rownames(matTraining)`.

```
> class(matTraining)

[1] "matrix" "array"

> dim(matTraining)

[1] 70 78

> str(matTraining)
```

```
 num [1:70, 1:78] -0.0564 0.0347 -0.0451 -0.1556 0.1394 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:70] "AA555029_RC_Hs.370457" "AF257175_Hs.15250" "AK000745_Hs.377155" "AKAP2_Hs.516834"
  ..$ : chr [1:78] "Training1.Bad" "Training2.Bad" "Training3.Good" "Training4.Good" ...
```

The factor `trainingGroup` contains the prognostic information:

```
> ### Show group variable for the TRAINING set
> table(trainingGroup)

trainingGroup
 Bad Good
  34   44
```

3.2 Testing set

Load the example testing data contained in the `switchBox` package.

```
> ### Load the example data for the TEST set
> data(testingData)
```

The object `matTesting` is a numeric matrix containing gene expression data for the 307 breast cancer patients and the 70 genes used to validate the MammaPrint assay [8]. This data was obtained from from the `MammaPrintData` package, as described in [1]. Also in this case samples are stored by column and genes by row. Gene annotation is stored as `rownames(matTraining)`.

```
> class(matTesting)

[1] "matrix" "array"

> dim(matTesting)

[1]  70 307

> str(matTesting)

 num [1:70, 1:307] 0.0035 -0.0599 -0.0678 0.1139 -0.094 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:70] "AA555029_RC_Hs.370457" "AF257175_Hs.15250" "AK000745_Hs.377155" "AKAP2_Hs.516834"
  ..$ : chr [1:307] "Test1.Good" "Test2.Good" "Test3.Good" "Test4.Good" ...
```

The factor `testingGroup` contains the prognostic information:

```
> ### Show group variable for the TEST set
> table(testingGroup)

testingGroup
 Bad Good
  47  260
```

4 Training KTSP algorithm

4.1 Unrestricted KTSP classifiers

We can train the KTSP algoritm using all possible feature pairs – unrestricted KTSP classifier – with or without statistical feature filtering, using the `SWAP.Train.KTSP` function.

Note that `SWAP.KTSP.Train` is deprecated and maintained only for legacy reasons.

4.1.1 Default statistical filtering

Training an unrestricted KTSP predictor using a statistical feature filtering is the default and it is achieved by using the default parameters, as follows:

```
> ### The arguments to the "SWAP.Train.KTSP" function
> args(SWAP.Train.KTSP)

function (inputMat, phenoGroup, classes = NULL, krange = 2:10,
    FilterFunc = SWAP.Filter.Wilcoxon, RestrictedPairs = NULL,
    handleTies = FALSE, disjoint = TRUE, k_selection_fn = KbyTtest,
    k_opts = list(), score_fn = signedTSPScores, score_opts = NULL,
    verbose = FALSE, ...)
NULL

> ### Train a classifier using default filtering function based on the Wilcoxon test
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup, krange=c(3:15))
> ### Show the classifier
> classifier

$name
[1] "7TSPS"

$TSPs
                                          gene1
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "GNAZ_Hs.555870"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "Contig46223_RC_Hs.22917"
RFC4_Hs.518475,L2DTL_Hs.445885            "RFC4_Hs.518475"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "Contig40831_RC_Hs.161160"
FLJ11354_Hs.523468,LOC57110_Hs.36761      "FLJ11354_Hs.523468"
Contig55725_RC_Hs.470654,IGFBP5_Hs.184339 "Contig55725_RC_Hs.470654"
UCH37_Hs.145469,SERF1A_Hs.32567           "UCH37_Hs.145469"
                                          gene2
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "Contig32185_RC_Hs.159422"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "OXCT_Hs.278277"
RFC4_Hs.518475,L2DTL_Hs.445885            "L2DTL_Hs.445885"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "CFFM4_Hs.250822"
FLJ11354_Hs.523468,LOC57110_Hs.36761      "LOC57110_Hs.36761"
Contig55725_RC_Hs.470654,IGFBP5_Hs.184339 "IGFBP5_Hs.184339"
UCH37_Hs.145469,SERF1A_Hs.32567           "SERF1A_Hs.32567"
```

```
$score
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                0.6029423                                 0.5467924
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                0.5347600                                 0.5280755
     FLJ11354_Hs.523468,LOC57110_Hs.36761 Contig55725_RC_Hs.470654,IGFBP5_Hs.184339
                                0.5267389                                 0.5200542
          UCH37_Hs.145469,SERF1A_Hs.32567
                                0.5133699

$tieVote
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                    both                                      both
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                    both                                      both
     FLJ11354_Hs.523468,LOC57110_Hs.36761 Contig55725_RC_Hs.470654,IGFBP5_Hs.184339
                                    both                                      both
          UCH37_Hs.145469,SERF1A_Hs.32567
                                    both
Levels: both Bad Good

$labels
[1] "Bad"  "Good"

> ### Extract the TSP from the classifier
> classifier$TSPs

                                          gene1
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "GNAZ_Hs.555870"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "Contig46223_RC_Hs.22917"
RFC4_Hs.518475,L2DTL_Hs.445885            "RFC4_Hs.518475"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "Contig40831_RC_Hs.161160"
FLJ11354_Hs.523468,LOC57110_Hs.36761      "FLJ11354_Hs.523468"
Contig55725_RC_Hs.470654,IGFBP5_Hs.184339 "Contig55725_RC_Hs.470654"
UCH37_Hs.145469,SERF1A_Hs.32567           "UCH37_Hs.145469"
                                          gene2
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "Contig32185_RC_Hs.159422"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "OXCT_Hs.278277"
RFC4_Hs.518475,L2DTL_Hs.445885            "L2DTL_Hs.445885"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "CFFM4_Hs.250822"
FLJ11354_Hs.523468,LOC57110_Hs.36761      "LOC57110_Hs.36761"
Contig55725_RC_Hs.470654,IGFBP5_Hs.184339 "IGFBP5_Hs.184339"
UCH37_Hs.145469,SERF1A_Hs.32567           "SERF1A_Hs.32567"
```

Below is shown the way the default feature filtering works. The `SWAP.Filter.Wilcoxon` function takes the phenotype factor, the predictor data, the number of feature to be returned, and a logical value to decide whether to include equal number of featured positively and negatively associated with the phenotype to be predicted.

```
> ### The arguments to the "SWAP.Train.KTSP" function
> args(SWAP.Filter.Wilcoxon)

function (phenoGroup, inputMat, featureNo = 100, UpDown = TRUE)
NULL
```

```
> ### Retrieve the top best 4 genes using default Wilcoxon filtering
> ### Note that there are ties
> SWAP.Filter.Wilcoxon(trainingGroup, matTraining, featureNo=4)

[1] "KIAA0175_Hs.184339" "IGFBP5_Hs.184339"   "RFC4_Hs.518475"
[4] "FLJ11354_Hs.523468" "GNAZ_Hs.555870"
```

Train a classifier using the `SWAP.Filter.Wilcoxon` filtering function.

```
> ### Train a classifier from the top 4 best genes
> ### according to Wilcoxon filtering function
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup,
                                FilterFunc=SWAP.Filter.Wilcoxon, featureNo=4)
> ### Show the classifier
> classifier

$name
[1] "2TSPS"

$TSPs
                                     gene1                gene2
IGFBP5_Hs.184339,FLJ11354_Hs.523468 "FLJ11354_Hs.523468" "IGFBP5_Hs.184339"
RFC4_Hs.518475,KIAA0175_Hs.184339    "RFC4_Hs.518475"     "KIAA0175_Hs.184339"

$score
IGFBP5_Hs.184339,FLJ11354_Hs.523468    RFC4_Hs.518475,KIAA0175_Hs.184339
                          0.5173798                            0.4826204

$tieVote
IGFBP5_Hs.184339,FLJ11354_Hs.523468    RFC4_Hs.518475,KIAA0175_Hs.184339
                                 both                                 both
Levels: both Bad Good

$labels
[1] "Bad"  "Good"
```

Train a classifier using all possible features:

```
> ### To use all features "FilterFunc" must be set to NULL
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup, FilterFunc=NULL)
> ### Show the classifier
> classifier

$name
[1] "7TSPS"

$TSPs
                                          gene1
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "GNAZ_Hs.555870"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "Contig46223_RC_Hs.22917"
RFC4_Hs.518475,L2DTL_Hs.445885            "RFC4_Hs.518475"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "Contig40831_RC_Hs.161160"
LOC57110_Hs.36761,FLJ11354_Hs.523468      "FLJ11354_Hs.523468"
IGFBP5_Hs.184339,Contig55725_RC_Hs.470654 "Contig55725_RC_Hs.470654"
```

```
UCH37_Hs.145469,SERF1A_Hs.32567           "UCH37_Hs.145469"
                                         gene2
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "Contig32185_RC_Hs.159422"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "OXCT_Hs.278277"
RFC4_Hs.518475,L2DTL_Hs.445885            "L2DTL_Hs.445885"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "CFFM4_Hs.250822"
LOC57110_Hs.36761,FLJ11354_Hs.523468      "LOC57110_Hs.36761"
IGFBP5_Hs.184339,Contig55725_RC_Hs.470654 "IGFBP5_Hs.184339"
UCH37_Hs.145469,SERF1A_Hs.32567           "SERF1A_Hs.32567"

$score
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                0.6029423                                 0.5467924
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                0.5347600                                 0.5280755
     LOC57110_Hs.36761,FLJ11354_Hs.523468 IGFBP5_Hs.184339,Contig55725_RC_Hs.470654
                                0.5267389                                 0.5200542
          UCH37_Hs.145469,SERF1A_Hs.32567
                                0.5133699

$tieVote
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                     both                                      both
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                     both                                      both
     LOC57110_Hs.36761,FLJ11354_Hs.523468 IGFBP5_Hs.184339,Contig55725_RC_Hs.470654
                                     both                                      both
          UCH37_Hs.145469,SERF1A_Hs.32567
                                     both
Levels: both Bad Good

$labels
[1] "Bad"  "Good"
```

4.1.2 Altenative filtering methods

Training can also be achieved using alternative filtering methods. These methods can be specified by passing a different filtering function to `SWAP.Train.KTSP`. These functions should use th `phenoGroup`, `inputData` arguments, as well as any other necessary argument (passed using `...`), as shown below.

For instance, we can define an alternative filtering function selecting 10 random features.

```
> ### An alternative filtering function selecting 20 random features
> random10 <- function(situation, data) { sample(rownames(data), 10) }
> random10(trainingGroup, matTraining)

 [1] "FGF18_Hs.87191"          "GNAZ_Hs.555870"          "SERF1A_Hs.32567"
 [4] "CEGP1_Hs.369982"         "TGFB3_Hs.2025"           "CENPA_Hs.1594"
 [7] "SM-20_Hs.444450"         "AP2B1_Hs.514819"         "Contig32125_RC_Hs.371395"
[10] "WISP1_Hs.492974"
```

Below is a more realistic example of an alternative filtering function. In this case we use the R `t.test` function to select the features with an absolute t-statistics larger than a specified quantile.

```
> ### An alternative filtering function based on a t-test
> topRttest <- function(situation, data, quant = 0.75) {
         out <- apply(data, 1, function(x, ...) t.test(x ~ situation)$statistic )
         names(out[ abs(out) > quantile(abs(out), quant) ])
 }
> ### Show the top 5% features using the newly defined filtering function
> topRttest(trainingGroup, matTraining, quant=0.95)

[1] "Contig32185_RC_Hs.159422" "FLJ11354_Hs.523468"       "IGFBP5_Hs.184339"
[4] "KIAA0175_Hs.184339"
```

Train a classifier using the alternative filtering function based on the t-test and also define the max number of TSP using `krange`.

```
> ### Train with t-test and krange
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup,
                                FilterFunc = topRttest, quant = 0.9, krange=c(15:30) )
> ### Show the classifier
> classifier

$name
[1] "15TSPS"

$TSPs
                                          gene1
Contig32185_RC_Hs.159422,GNAZ_Hs.555870 "GNAZ_Hs.555870"
IGFBP5_Hs.184339,FLJ11354_Hs.523468     "FLJ11354_Hs.523468"
SERF1A_Hs.32567,MMP9_Hs.297413          "SERF1A_Hs.32567"
KIAA0175_Hs.184339,KIAA0175_Hs.184339   "KIAA0175_Hs.184339"
NA                                      NA
NA.1                                    NA
NA.2                                    NA
NA.3                                    NA
NA.4                                    NA
NA.5                                    NA
NA.6                                    NA
NA.7                                    NA
NA.8                                    NA
NA.9                                    NA
NA.10                                   NA
                                          gene2
Contig32185_RC_Hs.159422,GNAZ_Hs.555870 "Contig32185_RC_Hs.159422"
IGFBP5_Hs.184339,FLJ11354_Hs.523468     "IGFBP5_Hs.184339"
SERF1A_Hs.32567,MMP9_Hs.297413          "MMP9_Hs.297413"
KIAA0175_Hs.184339,KIAA0175_Hs.184339   "KIAA0175_Hs.184339"
NA                                      NA
NA.1                                    NA
NA.2                                    NA
NA.3                                    NA
NA.4                                    NA
```

```
NA.5                                    NA
NA.6                                    NA
NA.7                                    NA
NA.8                                    NA
NA.9                                    NA
NA.10                                   NA

$score
Contig32185_RC_Hs.159422,GNAZ_Hs.555870     IGFBP5_Hs.184339,FLJ11354_Hs.523468
                              0.6029413                               0.5173798
         SERF1A_Hs.32567,MMP9_Hs.297413   KIAA0175_Hs.184339,KIAA0175_Hs.184339
                              0.1631016                               0.0000000
                                     NA                                    NA.1
                                     NA                                      NA
                                   NA.2                                    NA.3
                                     NA                                      NA
                                   NA.4                                    NA.5
                                     NA                                      NA
                                   NA.6                                    NA.7
                                     NA                                      NA
                                   NA.8                                    NA.9
                                     NA                                      NA
                                  NA.10
                                     NA

$tieVote
Contig32185_RC_Hs.159422,GNAZ_Hs.555870     IGFBP5_Hs.184339,FLJ11354_Hs.523468
                                  both                                    both
         SERF1A_Hs.32567,MMP9_Hs.297413   KIAA0175_Hs.184339,KIAA0175_Hs.184339
                                  both                                    both
                                     NA                                    NA.1
                                   <NA>                                    <NA>
                                   NA.2                                    NA.3
                                   <NA>                                    <NA>
                                   NA.4                                    NA.5
                                   <NA>                                    <NA>
                                   NA.6                                    NA.7
                                   <NA>                                    <NA>
                                   NA.8                                    NA.9
                                   <NA>                                    <NA>
                                  NA.10
                                   <NA>
Levels: both Bad Good

$labels
[1] "Bad"  "Good"
```

4.2 Training a Restricted KTSP algorithm

The `switncBox` allows to training a KTSP classifier using a pre-specified set of restricted feature pairs. This can be useful to implement KTSP classifiers restricted to specific TSPs based, for instane, on prior biological information ([9]).

To this end, the user must specify a set of candidate pairs by setting `RestrictedPairs` argument.

As an example, we can define a set of candidate pairs by randolmly selecting some of the rownames from the `inputMat` matrix and the classifier chooses from this set.

In a real example these pairs would be provided by the user, for instance usinf prior biological knowledge. The restricted pairs must contain valid feature names, *i.e.* the row names of `inputMat`.

```
> set.seed(123)
> somePairs <- matrix(sample(rownames(matTraining), 6^2, replace=FALSE), ncol=2)
> head(somePairs)

     [,1]                       [,2]
[1,] "DKFZP564D0462_Hs.318894" "Contig63649_RC_Hs.72620"
[2,] "LOC51203_Hs.511093"      "BBC3_Hs.467020"
[3,] "COL4A2_Hs.508716"        "DC13_Hs.388255"
[4,] "TGFB3_Hs.2025"           "FGF18_Hs.87191"
[5,] "GNAZ_Hs.555870"          "AP2B1_Hs.514819"
[6,] "L2DTL_Hs.445885"         "ORC6L_Hs.49760"

> dim(somePairs)

[1] 18  2
```

Train a classifier using the set of restricted feature pairs and the default filtering:

```
> ### Train
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup,
                                RestrictedPairs = somePairs, krange=3:16)
> ### Show the classifier
> classifier

$name
[1] "4TSPS"

$TSPs
                                    gene1                gene2
UCH37_Hs.145469,ECT2_Hs.518299   "UCH37_Hs.145469"    "ECT2_Hs.518299"
SERF1A_Hs.32567,PK428_Hs.516834  "PK428_Hs.516834"    "SERF1A_Hs.32567"
GNAZ_Hs.555870,AP2B1_Hs.514819   "GNAZ_Hs.555870"     "AP2B1_Hs.514819"
FLJ11354_Hs.523468,PECI_Hs.15250 "FLJ11354_Hs.523468" "PECI_Hs.15250"

$score
  UCH37_Hs.145469,ECT2_Hs.518299  SERF1A_Hs.32567,PK428_Hs.516834
                       0.4719255                        0.3836903
  GNAZ_Hs.555870,AP2B1_Hs.514819 FLJ11354_Hs.523468,PECI_Hs.15250
                       0.3743319                        0.3609628

$tieVote
  UCH37_Hs.145469,ECT2_Hs.518299  SERF1A_Hs.32567,PK428_Hs.516834
```

```
                              both                                   both
  GNAZ_Hs.555870,AP2B1_Hs.514819 FLJ11354_Hs.523468,PECI_Hs.15250
                              both                                   both
Levels: both Bad Good

$labels
[1] "Bad"  "Good"
```

Train a classifier using a set of restricted feature pairs, defining the maximum number of TSP using `krange` and also filtering the features by T-test.

```
> ### Train
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup,
                                RestrictedPairs = somePairs,
                                FilterFunc = topRttest, quant = 0.3,
                                krange=c(3:10) )
> ### Show the classifier
> classifier

$name
[1] "3TSPS"

$TSPs
                                  gene1                gene2
UCH37_Hs.145469,ECT2_Hs.518299   "UCH37_Hs.145469"    "ECT2_Hs.518299"
SERF1A_Hs.32567,PK428_Hs.516834  "PK428_Hs.516834"    "SERF1A_Hs.32567"
FLJ11354_Hs.523468,PECI_Hs.15250 "FLJ11354_Hs.523468" "PECI_Hs.15250"

$score
  UCH37_Hs.145469,ECT2_Hs.518299  SERF1A_Hs.32567,PK428_Hs.516834
                       0.4719254                        0.3836902
FLJ11354_Hs.523468,PECI_Hs.15250
                       0.3609628

$tieVote
  UCH37_Hs.145469,ECT2_Hs.518299  SERF1A_Hs.32567,PK428_Hs.516834
                              both                             both
FLJ11354_Hs.523468,PECI_Hs.15250
                              both
Levels: both Bad Good

$labels
[1] "Bad"  "Good"
```

5 Calculate and aggregates the TSP votes

The `SWAP.KTSP.Statistics` function can be used to compute and aggregate the TSP votes using alternative functions to combine the votes. The default method is the count of the signed TSP votes. We can also pass a different function

to combine the KTSPs. This function takes an argument `x` – a logical vector corresponding to the TSP votes – of length equal to the number of columns (*e.g.*, the number of cancer patients under analysis) and aggregates the votes of all K TSPs of the classifier identified by the training proces (see the `SWAP.Train.KTSP` function).

Here we will use the default parameters (the count of the signed TSP votes)

```
> ### Train a classifier
> classifier <- SWAP.Train.KTSP(matTraining, trainingGroup,
                                FilterFunc = NULL, krange=2:8)
> ### Compute the statistics using the default parameters:
> ### counting the signed TSP votes
> ktspStatDefault <- SWAP.KTSP.Statistics(inputMat = matTraining,
     classifier = classifier)
> ### Show the components in the output
> names(ktspStatDefault)

[1] "statistics"  "comparisons"

> ### Show some of the votes
> head(ktspStatDefault$comparisons[ , 1:2])

               GNAZ_Hs.555870>Contig32185_RC_Hs.159422
Training1.Bad                                   FALSE
Training2.Bad                                   FALSE
Training3.Good                                   TRUE
Training4.Good                                   TRUE
Training5.Bad                                   FALSE
Training6.Bad                                   FALSE
               Contig46223_RC_Hs.22917>OXCT_Hs.278277
Training1.Bad                                  FALSE
Training2.Bad                                  FALSE
Training3.Good                                  TRUE
Training4.Good                                  TRUE
Training5.Bad                                   TRUE
Training6.Bad                                  FALSE

> ### Show default statistics
> head(ktspStatDefault$statistics)

 Training1.Bad  Training2.Bad Training3.Good Training4.Good  Training5.Bad
          -2.5           -2.5            2.5            2.5           -0.5
 Training6.Bad
          -0.5
```

Here we will use the sum to aggregate the TSP votes

```
> ### Compute
> ktspStatSum <- SWAP.KTSP.Statistics(inputMat = matTraining,
     classifier = classifier, CombineFunc=sum)
> ### Show statistics obtained using the sum
> head(ktspStatSum$statistics)
```

```
 Training1.Bad  Training2.Bad Training3.Good Training4.Good  Training5.Bad
             1              1              6              6              3
Training6.Bad
             3
```

Here, for instance, we will apply a hard treshold equal to 2

```
> ### Compute
> ktspStatThreshold <- SWAP.KTSP.Statistics(inputMat = matTraining,
     classifier = classifier,  CombineFunc = function(x) sum(x) > 2 )
> ### Show statistics obtained using the threshold
> head(ktspStatThreshold$statistics)

 Training1.Bad  Training2.Bad Training3.Good Training4.Good  Training5.Bad
         FALSE          FALSE           TRUE           TRUE           TRUE
Training6.Bad
          TRUE
```

We can also make a heatmap showing the individual TSPs votes (see Figure 1 below).

```
> ### Make a heatmap showing the individual TSPs votes
> colorForRows <- as.character(1+as.numeric(trainingGroup))
> heatmap(1*ktspStatThreshold$comparisons, scale="none",
     margins = c(10, 5), cexCol=0.5, cexRow=0.5,
     labRow=trainingGroup, RowSideColors=colorForRows)
```

6 Classifiy samples and compute the classifier performance

6.1 Classifiy training samples

The `SWAP.KTSP.Classify` function allows to classify one or more samples using the classifier identified by `SWAP.Train.KTSP`. The **resubstitution** performance in the training set is shown below.

```
> ### Show the classifier
> classifier

$name
[1] "7TSPS"

$TSPs
                                          gene1
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "GNAZ_Hs.555870"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "Contig46223_RC_Hs.22917"
RFC4_Hs.518475,L2DTL_Hs.445885            "RFC4_Hs.518475"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "Contig40831_RC_Hs.161160"
LOC57110_Hs.36761,FLJ11354_Hs.523468      "FLJ11354_Hs.523468"
IGFBP5_Hs.184339,Contig55725_RC_Hs.470654 "Contig55725_RC_Hs.470654"
UCH37_Hs.145469,SERF1A_Hs.32567           "UCH37_Hs.145469"
                                          gene2
Contig32185_RC_Hs.159422,GNAZ_Hs.555870   "Contig32185_RC_Hs.159422"
Contig46223_RC_Hs.22917,OXCT_Hs.278277    "OXCT_Hs.278277"
RFC4_Hs.518475,L2DTL_Hs.445885            "L2DTL_Hs.445885"
Contig40831_RC_Hs.161160,CFFM4_Hs.250822  "CFFM4_Hs.250822"
LOC57110_Hs.36761,FLJ11354_Hs.523468      "LOC57110_Hs.36761"
IGFBP5_Hs.184339,Contig55725_RC_Hs.470654 "IGFBP5_Hs.184339"
UCH37_Hs.145469,SERF1A_Hs.32567           "SERF1A_Hs.32567"

$score
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                0.6029423                                 0.5467924
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                0.5347600                                 0.5280755
     LOC57110_Hs.36761,FLJ11354_Hs.523468 IGFBP5_Hs.184339,Contig55725_RC_Hs.470654
                                0.5267389                                 0.5200542
          UCH37_Hs.145469,SERF1A_Hs.32567
                                0.5133699

$tieVote
  Contig32185_RC_Hs.159422,GNAZ_Hs.555870    Contig46223_RC_Hs.22917,OXCT_Hs.278277
                                     both                                      both
           RFC4_Hs.518475,L2DTL_Hs.445885  Contig40831_RC_Hs.161160,CFFM4_Hs.250822
                                     both                                      both
     LOC57110_Hs.36761,FLJ11354_Hs.523468 IGFBP5_Hs.184339,Contig55725_RC_Hs.470654
                                     both                                      both
          UCH37_Hs.145469,SERF1A_Hs.32567
                                     both
```

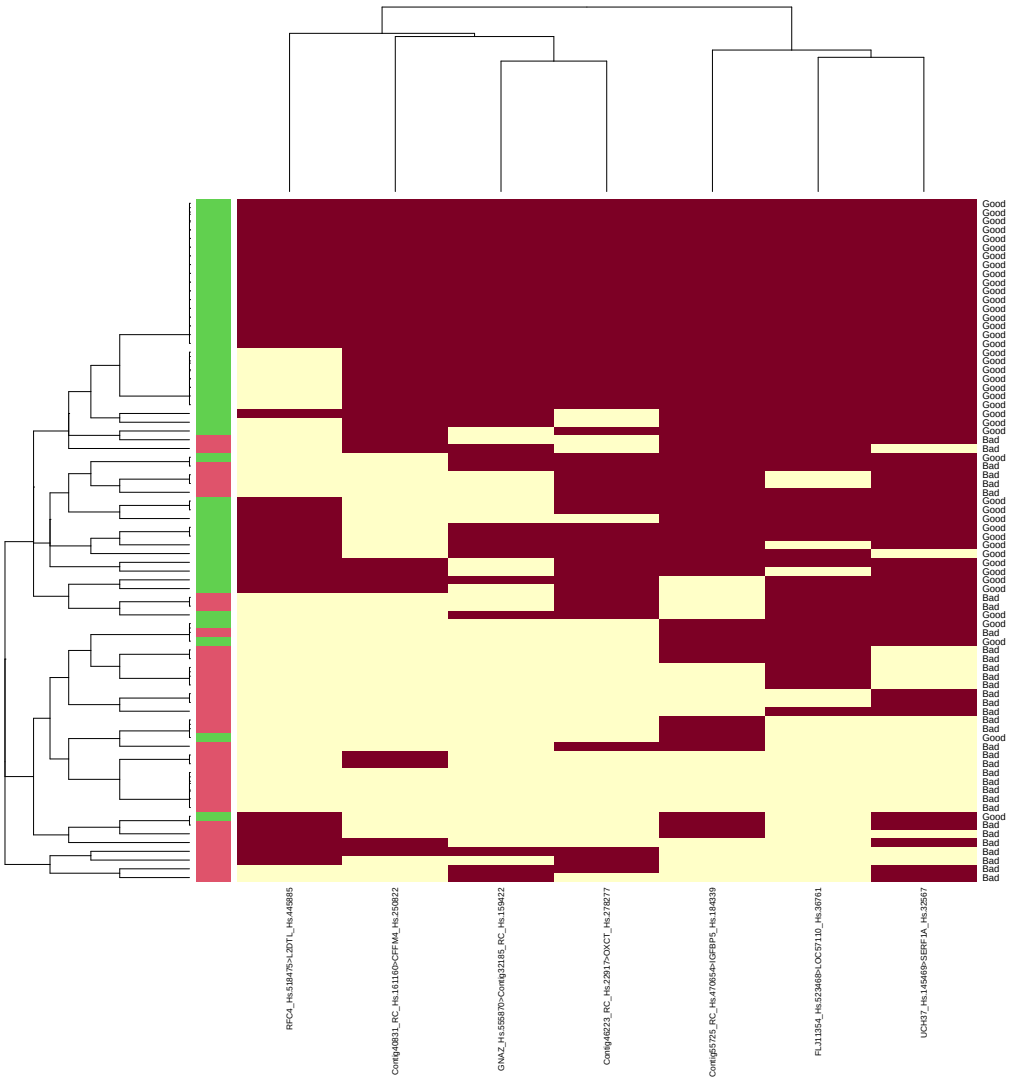


Figure 1: Heatmap showing the individual TSP votes.

```
Levels: both Bad Good

$labels
[1] "Bad"  "Good"

> ### Apply the classifier to the TRAINING set
> trainingPrediction <- SWAP.KTSP.Classify(matTraining, classifier)
> ### Show
> str(trainingPrediction)

 Factor w/ 2 levels "Bad","Good": 1 1 2 2 1 1 2 2 1 1 ...
 - attr(*, "names")= chr [1:78] "Training1.Bad" "Training2.Bad" "Training3.Good" "Training4.Good" ...

> ### Resubstitution performance in the TRAINING set
> table(trainingPrediction, trainingGroup)

                  trainingGroup
trainingPrediction Bad Good
              Bad   29    4
              Good   5   40
```

We can apply the classifier using a specific decision to combine the K TSP as specified with the `DecideFunc` argument of `SWAP.KTSP.Classify`. This argument is a function working on a logical vector `x` containing the votes of each TSP. We can for instance count all votes for class one and then classify a patient in one class or the other based on a specific threshold.

```
> ### Usr a CombineFunc based on  sum(x) > 5.5
> trainingPrediction <- SWAP.KTSP.Classify(matTraining, classifier,
                                           DecisionFunc = function(x) sum(x) > 5.5 )
> ### Show
> str(trainingPrediction)

 Factor w/ 2 levels "Bad","Good": 1 1 2 2 1 1 1 2 1 1 ...
 - attr(*, "names")= chr [1:78] "Training1.Bad" "Training2.Bad" "Training3.Good" "Training4.Good" ...

> ### Resubstitution performance in the TRAINING set
> table(trainingPrediction, trainingGroup)

                  trainingGroup
trainingPrediction Bad Good
              Bad   34   15
              Good   0   29
```

6.2 Classifiy validation samples

We can apply the trained classifier to one new sample of the test set:

```
> ### Classify one sample
> testPrediction <- SWAP.KTSP.Classify(matTesting[ , 1, drop=FALSE], classifier)
> ### Show
> testPrediction

Test1.Good
      Good
Levels: Bad Good
```

We can apply the trained classifier to a new set of samples, using the defaul decision rule based on the "majority wins" principle:

```
> ### Apply the classifier to the complete TEST set
> testPrediction <- SWAP.KTSP.Classify(matTesting, classifier)
> ### Show
> table(testPrediction)

testPrediction
 Bad Good
 108  199

> ### Resubstitution performance in the TEST set
> table(testPrediction, testingGroup)

              testingGroup
testPrediction Bad Good
          Bad   27   81
          Good  20  179
```

We can apply the trained classifier to predict of a new set of samples, using an alternative decision rule specified by `DecideFunc` For instance, we can classify by thresholding vote counts in favor of one of the classes.

```
> ### APlly the classifier using sum(x)  > 5.5
> testPrediction <- SWAP.KTSP.Classify(matTesting, classifier,
                                       DecisionFunc = function(x) sum(x) > 5.5 )
> ### Resubstitution performance in the TEST set
> table(testPrediction, testingGroup)

              testingGroup
testPrediction Bad Good
          Bad   44  163
          Good   3   97
```

7 Compute the signed TSP scores

The `switchBox` allows also to compute the individual scores for each TSP of interest. This can be achieved by using the `SWAP.CalculateSignedScore` function as shown below.

Compute the scores using all features for all possible pairs:

```
> ### Compute the scores using all features for all possible pairs
> scores <- SWAP.CalculateSignedScore(matTraining,  trainingGroup, FilterFunc=NULL)
> ### Show scores
> class(scores)

[1] "list"

> dim(scores$score)

[1] 70 70
```

Extract the TSP scores of interest – the absolute value correspond to the scores returned by `SWAP.Train.KTSP`.

```
> ### Get the scores
> scoresOfInterest <- diag(scores$score[ classifier$TSPs[,1] , classifier$TSPs[,2] ])
> ### Their absolute value should corresponf to the scores returned by SWAP.Train.KTSP
> all(classifier$score == abs(scoresOfInterest))

[1] FALSE
```

The `SWAP.CalculateSignedScore` function accept the same argumets used by `SWAP.Train.KTSP`. It can compute the scores with or without a filtering function and using or not the restricted pairs, as specified by `FilterFunc` and `RestrictedPairs` respectively.

```
> ### Compute the scores with default filtering function
> scores <- SWAP.CalculateSignedScore(matTraining, trainingGroup, featureNo=20 )
> ### Show scores
> dim(scores$score)

[1] 21 21

> ### Compute the scores without the default filtering function
> ### and using restricted pairs
> scores <- SWAP.CalculateSignedScore(matTraining, trainingGroup,
                                      FilterFunc = NULL, RestrictedPairs = somePairs )
> ### Show scores
> class(scores$score)

[1] "numeric"

> length(scores$score)

[1] 18
```

In Figure 2 is shown the histograms for all possible TSP scores.

```
> hist(scores$score, col="salmon", main="TSP scores")
```

```
> hist(scores$score, col="salmon", main="TSP scores")
```

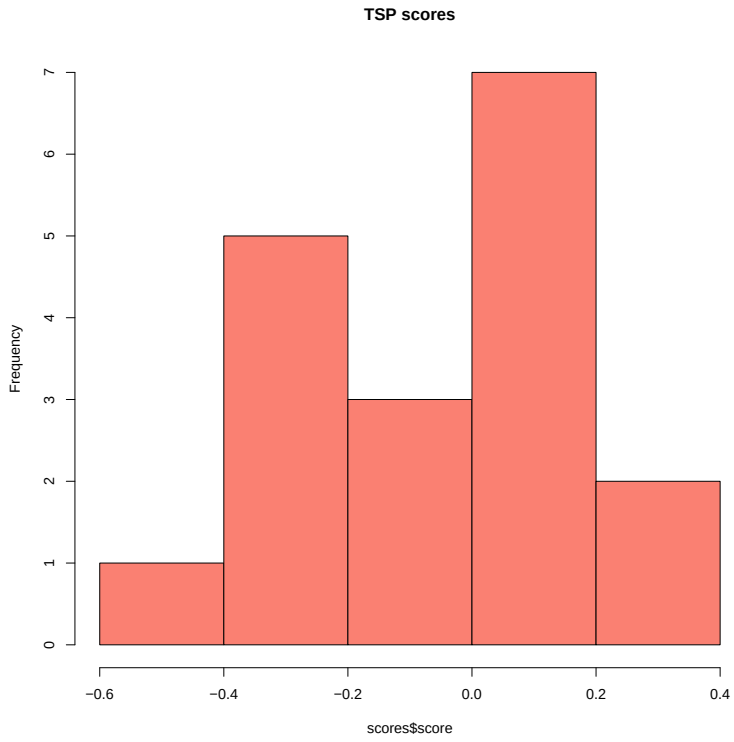


Figure 2: Histograms of all TSP socres.

8 Use of deprecated functions

The two functions `KTSP.Train` and `KTSP.Classify` are deprecated and are included in the package only for backward compatibility. They have been substituted by respectively `SWAP.Train.KTSP` and `SWAP.KTSP.Classify`. These functions were used to train and validate the 8-TSP classifier described by Marchionni et al [1] and are maintained for reproducibility purposes. Example on the way they are used follows.

Preparation of phenotype information (a numeric vector with values equal to 0 or 1) for training the KTSP classifier:

```
> ### Phenotypic group variable for the 78 samples
> table(trainingGroup)

trainingGroup
 Bad Good
  34   44

> levels(trainingGroup)

[1] "Bad"  "Good"

> ### Turn into a numeric vector with values equal to 0 and 1
> trainingGroupNum <- as.numeric(trainingGroup) - 1
> ### Show group variable for the TRAINING set
> table(trainingGroupNum)

trainingGroupNum
 0  1
34 44
```

KTSP classifier training using the deprected function:

```
> ### Train a classifier using default filtering function based on the Wilcoxon test
> classifier <- KTSP.Train(matTraining, trainingGroupNum, n=8)
> ### Show the classifier
> classifier

$TSPs
     [,1] [,2]
[1,]   42   19
[2,]   24   58
[3,]   63   50
[4,]   22   13
[5,]   37   52
[6,]   27   46
[7,]   69   64
[8,]   43   48

$score
[1] 0.6029417 0.5467919 0.5347597 0.5280752 0.5267384 0.5200538 0.5133694 0.5080218
```

```
$geneNames
     [,1]                       [,2]
[1,] "GNAZ_Hs.555870"           "Contig32185_RC_Hs.159422"
[2,] "Contig46223_RC_Hs.22917"  "OXCT_Hs.278277"
[3,] "RFC4_Hs.518475"           "L2DTL_Hs.445885"
[4,] "Contig40831_RC_Hs.161160" "CFFM4_Hs.250822"
[5,] "FLJ11354_Hs.523468"       "LOC57110_Hs.36761"
[6,] "Contig55725_RC_Hs.470654" "IGFBP5_Hs.184339"
[7,] "UCH37_Hs.145469"          "SERF1A_Hs.32567"
[8,] "GSTM3_Hs.2006"            "KIAA0175_Hs.184339"
```

KTSP classifier performance using the deprected function:

```
> ### Apply the classifier to one sample of the TEST set using
> ### sum of votes less  than 2.5
> trainPrediction <- KTSP.Classify(matTraining, classifier,
                                   combineFunc = function(x) sum(x) < 2.5)
> ### Contingency table
> table(trainPrediction, trainingGroupNum)

               trainingGroupNum
trainPrediction  0  1
              0 34  8
              1  0 36
```

Preparation of phenotype information (a numeric vector with values equal to 0 or 1) for testing the KTSP classifier on new data:

```
> ### Phenotypic group variable for the 307 samples
> table(testingGroup)

testingGroup
 Bad Good
  47  260

> levels(testingGroup)

[1] "Bad"  "Good"

> ### Turn into a numeric vector with values equal to 0 and 1
> testingGroupNum <- as.numeric(testingGroup) - 1
> ### Show group variable for the TEST set
> table(testingGroupNum)

testingGroupNum
  0   1
 47 260
```

Testing on new data and getting KTSP classifier performance using the deprected function:

```
> ### Apply the classifier to one sample of the TEST set using
> ### sum of votes less than 2.5
> testPrediction <- KTSP.Classify(matTesting, classifier,
      combineFunc = function(x) sum(x) < 2.5)
> ### Show prediction
> table(testPrediction)

testPrediction
  0   1
181 126

> ### Contingency table
> table(testPrediction, testingGroupNum)

              testingGroupNum
testPrediction   0   1
             0  43 138
             1   4 122
```

9 System Information

Session information:

```
> toLatex(sessionInfo())
```

- R version 4.6.1 (2026-06-24), x86_64-pc-linux-gnu
- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_GB, LC_COLLATE=C, LC_MONETARY=en_US.UTF-8, LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C
- Time zone: America/New_York
- TZcode source: system (glibc)
- Running under: Ubuntu 24.04.5 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.24-bioc/R/lib/libRblas.so
- LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: gplots 3.3.0, pROC 1.19.1, switchBox 1.50.0
- Loaded via a namespace (and not attached): KernSmooth 2.23-27, Rcpp 1.1.2, bitops 1.1-0, caTools 1.18.4, compiler 4.6.1, gtools 3.9.5, tools 4.6.1

10 Literature Cited

References

[1] Luigi Marchionni, Bahman Afsari, Donald Geman, and Jeffrey T Leek. A simple and reproducible breast cancer prognostic test. *BMC Genomics*, 14:336, 2013.

[2] Donald Geman, Christian d'Avignon, Daniel Q Naiman, and Raimond L Winslow. Classifying gene expression profiles from pairwise mrna comparisons. *Stat Appl Genet Mol Biol*, 3:Article19, 2004.

[3] Aik Choon Tan, Daniel Q Naiman, Lei Xu, Raimond L Winslow, and Donald Geman. Simple decision rules for classifying human cancers from gene expression profiles. *Bioinformatics*, 21(20):3896–904, Oct 2005.

[4] Lei Xu, Aik Choon Tan, Daniel Q Naiman, Donald Geman, and Raimond L Winslow. Robust prostate cancer marker genes emerge from direct integration of inter-study microarray data. *Bioinformatics*, 21(20):3905–11, Oct 2005.

[5] D. Geman, B. Afsari, and D. Naiman A.C. Tan. Microarray classification from several two-gene experssion comparisons. 2008. (Winner, ICMLA Microarray Classification Algorithm Competition).

[6] Bahman Afsari, Ulissess Braga-Neto, and Donald Geman. Rank discriminants for predicting phenotypes from rna expression. *Annals of Applied Statistics*, to appear.

[7] Annuska M Glas, Arno Floore, Leonie J M J Delahaye, Anke T Witteveen, Rob C F Pover, Niels Bakx, Jaana S T Lahti-Domenici, Tako J Bruinsma, Marc O Warmoes, René Bernards, Lodewyk F A Wessels, and Laura J Van't Veer. Converting a breast cancer microarray signature into a high-throughput diagnostic test. *BMC Genomics*, 7:278, 2006.

[8] Marc Buyse, Sherene Loi, Laura van't Veer, Giuseppe Viale, Mauro Delorenzi, Annuska M Glas, Mahasti Saghatchian d'Assignies, Jonas Bergh, Rosette Lidereau, Paul Ellis, Adrian Harris, Jan Bogaerts, Patrick Therasse, Arno Floore, Mohamed Amakrane, Fanny Piette, Emiel Rutgers, Christos Sotiriou, Fatima Cardoso, Martine J Piccart, and TRANSBIG Consortium.

Validation and clinical utility of a 70-gene prognostic signature for women with node-negative breast cancer. *J Natl Cancer Inst*, 98(17):1183–92, Sep 2006.

[9] Relative mRNA Levels of Functionally Interacting Proteins Are Consistent Disease Molecular Signatures. Wang, yuliang and afsari, bahman and geman, donald and price, nathan. *PLOS ONE*, Under revision.