Checking gene expression signatures against random and known signatures with *SigCheck*

Justin Norden and Rory Stark

Edited: October 3, 2014; Compiled: March 19, 2015

Contents

1 Introduction

A common task in the analysis of genomic data is the derivation of gene expression signatures that distinguish between phenotypes (disease outcomes, molecular subtypes, etc.). However, in their paper "Most random gene expression signatures are significantly associated with breast cancer outcome" [1], Venet, Dumont, and Detour point out that while a gene signature may distinguish between two classes of phenotype, their ultimate uniqueness and utility may be limited. They show that while a specialized feature selection process may appear to determine a unique set of predictor genes, the resultant signature may not perform better than one made up of random genes, or genes selected at random from all differentially expressed genes. This suggests that the genes in the derived signature may not be particularly informative as to underlying biological mechanisms. They further show that gene sets that comprise published signatures for a wide variety of phenotypic classes may perform just as well at predicting arbitrary phenotypes; famously, they show that a gene signature that distinguishes postprandial laughter performs as well at predicting the outcome of breast cancers as well as a widely-cited signature [2].

The *SigCheck* package was developed in order to make it easy to check a gene signature against random and known signatures, and assess the unique ability of that signature. It additionally provides the ability to check a signature's performance against permuted data as a reality check that it is detecting a genuine signal in the original data. This vignette shows the process of performing the checks.

2 Example: NKI Breast Cancer Data and the van't Veer Signature

In order to use *SigCheck*, you must provide a) some data (an expression matrix), b) some metadata (feature names and class identifiers), and c) a gene signature (a subset of the features). For this vignette, we will use the NKI Breast Cancer dataset *breastCancerNKI* and the associated van't Veer signature that predicts the likelihood that a patient will develop a distant metastases [2].

The dataset can be loaded as follows:

```
> library(breastCancerNKI)
> data(nki)
> nki
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 24481 features, 337 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: NKI_4 NKI_6 ... NKI_404 (337 total)
  varLabels: samplename dataset ... e.os (21 total)
  varMetadata: labelDescription
featureData
  featureNames: Contig45645_RC Contig44916_RC ... Contig15167_RC (24481 total)
  fvarLabels: probe EntrezGene.ID ... Description (10 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation: rosetta
```

As can be seen, the `nki` data is encapsulated in an `ExpressionSet` object. At its core, it contains an expression matrix consisting of 24,481 features (microarray probes mapped to genes) and 337 samples (derived from tumor tissue taken from breast cancer patients). Included is phenotypical (clinical) metadata regarding the patients, including age of the patient, tumor grade, expression status of the ER, PR, and HER2 biomarkers, presence of a BRCA mutation, etc:

```
> varLabels(nki)
```

```
 [1] "samplename"   "dataset"      "series"       "id"           "filename"
 [6] "size"         "age"          "er"           "grade"        "pgr"
[11] "her2"         "brca.mutation" "e.dmfs"       "t.dmfs"       "node"
[16] "t.rfs"        "e.rfs"        "treatment"    "tissue"       "t.os"
[21] "e.os"
```

The signatures derived by [2] and [3] are used to predict the Distant Metastasis-Free Survival (DMFS) time. This is presented as a continuous measure indicating the time until the occurrence of a distant metastasis:

```
> nki$t.dmfs
```

```
  [1] 4747 4075 3703 3215 3760 2120 2870 2983 3007 2873 1898 3227 2546 1894 2281 4160 4322
 [18] 4646 4424 3734 3789 3692 2675 2686 4024 2486 2482 1997 2150 1891  398  375 1792 1715
 [35]  839  410 1691 1787  979  295  748  978  724 1106  785  807  777 1347  843   NA   NA
 [52]   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA 1201
 [69] 1809  672  701  929  325 1167  792  464  364 4537 3117 1937 1911 3643 3688 5412 5209
 [86] 2427 2830 2308 1703 3192 2764 2665  407 2143 3159 2555 3408 1256 5599 1269 4663 2029
[103]  512 5528 5160 2004 1335  588 6699 6297  351 5118  430 5517  340 6450 2876 1027 1624
[120] 5898 2969 5593 5778 2069 3814  589 5597 5437 4876  507 5022 2774 4592 3260 4812 4662
[137]  955 3951 4134 4332  442 1950 4288 4567 4114 4410 2721 4652  985 4322 4553  749 4089
[154] 4035 3411 3984 3933 4091 1234 4031 3703 3526 3899 2398 4092 4436  721 2731 3781  627
[171]  855 3591 3772 3790  823 3660  972 1226  447  591   99 1308 5158 2380  907  421  674
[188] 2956 1496  732 3646 4217 1526 2059 1802 1942 2034 3436 3332 1676 3283  839 1869 2015
```

```
[205] 3033 3139  812 2649 2480 2561 2531 2589  342   94 2565 2649 2556 2164   20  237 1868
[222] 1940    9 2681 2098 1945 1430 2108 1806 2216  129 1816 4256 3056 2306 2244 2029  119
[239] 3505 3454 1042 3408  651 3358 2451 3488 3726  718 3405 3127  970 1541 3325 2212 1176
[256] 3010  781  853 2327 3614  548 2449 3215 3236 3234 3031 1688 2037  778 1899  788 2919
[273] 3103 2810 2731 2706  762 2317 6060 1140  633 5607 2414 2511 2555 2601 1724 2254 2361
[290] 1200 2384 2122 1769 2250 2208 2270 2127 2279 2198 2027 1953 1921 1816 6604 6387 6265
[307]  209 3495 1190 3652  719 2839  979 6363 3115 5084 5064 4653 2839 4048  711 3000 2639
[324] 1249 2485 1282 2254 2036 2085 4095 3737 1741 3077  558 2695 2467 2765
```

DMFS is also presented as a binary class, with a cutoff used to distinguish between patients that had a recurrence event and those that did not:

```
> nki$e.dmfs
```

```
  [1]  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 [29]  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA NA NA NA NA NA NA
 [57] NA NA NA NA NA NA NA NA NA NA NA  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  0  0
 [85]  0  0  0  1  1  0  0  0  0  0  0  1  0  1  0  1  0  0  1  0  0  0  1  1  0  0  1  1
[113]  1  0  1  0  0  1  1  0  1  0  0  0  1  1  0  0  0  1  0  1  0  1  0  0  1  0  0  0
[141]  1  0  0  0  0  0  0  0  1  0  1  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0  1  1
[169]  0  1  1  0  0  0  1  0  0  1  1  1  1  1  0  0  0  1  1  0  1  1  0  0  0  0  0  0
[197]  0  0  1  1  1  1  1  1  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0
[225]  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0  0  1  0  0  1  0  1  0  1
[253]  0  1  1  0  1  1  1  0  1  0  0  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0  1
[281]  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  1  1
[309]  1  1  1  0  1  0  1  0  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0  1  0  1  0  0
[337]  0
```

While the derived signatures were used as part of a survival analysis using the continuous measure, *SigCheck* uses a pure classification model and hence relies on the binary measure instead. A number of patient samples do not have DMFS data available, so the first step is to exclude these patients from our analysis, which brings the number of samples down to 319:

```
> dim(nki)
```

```
Features  Samples
   24481      337
```

```
> nki <- nki[,!is.na(nki$e.dmfs)]
> dim(nki)
```

```
Features  Samples
   24481      319
```

The next step is to provide a gene signature to check. A core function of *SigCheck* is to compare the performance of a gene signature with the performance of known gene signatures against the same data set. To accomplish this, it includes several sets of known signatures. One of these included signatures is the van't Veer signature, which we will use for this example.

To load the known gene signatures that are included with *SigCheck*:

```
> library(SigCheck)
> data(knownSignatures)
> names(knownSignatures)
```

```
[1] "cancer"        "proliferation" "non.cancer"
```

There are five sets of gene signatures, including a set of cancer signatures. The van't Veer signature is one of the signatures in the known cancer gene signature set:

```
> names(knownSignatures$cancer)
```

```
 [1] "ABBA"            "ADORNO"          "BEN-PORATH-EXP1" "BEN-PORATH-PRC2"
 [5] "BUESS"           "BUFFA"           "CARTER"          "CHANG"
 [9] "CHI"             "CRAWFORD"        "DAI"             "GLINSKY"
[13] "HALLSTROM"       "HE"              "HU"              "HUA"
[17] "IVSHINA"         "KOK"             "KORKOLA"         "LIU"
[21] "MA"              "MILLER"          "MORI"            "PAIK"
[25] "PAWITAN"         "PEI"             "RAMASWAMY"       "REUTER"
[29] "RHODES"          "SAAL"            "SHIPITSIN"       "SORLIE"
[33] "SOTIRIOU-93"     "SOTIRIOU-GGI"    "META-PCNA"       "TAUBE"
[37] "TAVAZOIE"        "VALASTYAN"       "VANTVEER"        "WANG-76"
[41] "WANG-ALK5T204D"  "WELM"            "WEST"            "WHITFIELD"
[45] "WONG-ESC"        "WONG-MITOCHON"   "WONG-PROTEAS"    "YU"
```

```
> vantveer <- knownSignatures$cancer$VANTVEER
> vantveer
```

```
 [1] "ACADS"    "AP2B1"    "ASNS"     "BUB1"     "CA9"      "CENPA"    "COL4A2"
 [8] "CP"       "DCK"      "ECT2"     "EXT1"     "FLT1"     "GNAZ"     "GSTM3"
[15] "IGFBP5"   "MCM6"     "MMP9"     "ALDH6A1"  "OXCT1"    "PEX12"    "PGK1"
[22] "QDPR"     "RAD21"    "RFC4"     "SLC2A3"   "STK3"     "TGFB3"    "BTG2"
[29] "SERF1A"   "CDC42BPA" "TMEFF1"   "FGF18"    "GMPS"     "WISP1"    "PRC1"
[36] "CCNB2"    "CCNE2"    "MELK"     "NDC80"    "PECI"     "PITRM1"   "NMU"
[43] "GCN1L1"   "ESM1"     "AKAP2"    "ORC6L"    "BBC3"     "UCHL5"    "DTL"
[50] "RAB6B"    "EGLN1"    "C20orf46" "STK32B"   "DEPDC1"   "C1orf106" "C16orf61"
[57] "ERGIC1"   "SCUBE2"   "MS4A7"    "FBXO31"
```

The signature is provided in the form of symbolic gene names. These will need to be matched up with the feature names in the `ExpressionSet`. While the default annotation is a probe identifier, the `nki` dataset provides a number of alternative annotations:

```
> fvarLabels(nki)
```

```
 [1] "probe"              "EntrezGene.ID"      "probe.name"
 [4] "Alignment.score"    "Length.of.probe"    "NCBI.gene.symbol"
 [7] "HUGO.gene.symbol"   "Cytoband"           "Alternative.symbols"
[10] "Description"
```

We'll be using the `"HUGO.gene-symbol"` to match the gene names in the van't Veer signature.

2.1 Running with a validation set

Optimally, when deriving and testing a gene signature, there are distinct sets of training samples and validation samples, with the validation samples kept separate from the process used to derive the gene signature. For this example, we are going to assume that the final 45 samples comprise the validation set. With this assumption, all required elements have been identified for an analysis.

SigCheck provides a high-level function, `sigCheck`, that will run all the checks and plot the results. This function accepts the expression and metadata in the form of an `ExpressionSet`; an indication of which metadata field should be used to define the two classes and which should be used as the annotation for the features; a gene signature to check; and an indication of which samples should be treated as the validation set. There are additional optional parameters that default to sensible values. For example, `sigCheck` also accepts the number of iterations to run for the random and permuted checks; this is set to a low number (10) by default to ensure a quick run, but in general should be set higher for more reliable p-values.

```
> nkiResults <- sigCheck(nki, classes="e.dmfs", annotation="HUGO.gene.symbol",
+                     signature=vantveer, validationSamples=275:319,
+                     knownSignatures="cancer",nIterations=1000)
```

Rather than run the 1,000 iterations in the vignette, a results list is included for a run of sigCheck with nIterations=1000:

```
> data(nkiResults)
```

sigCheck generates four plots, each showing how the baseline and mode performance compares to the distribution of results for a specific check. The results are best interpreted by looking more closely at what happens when sigCheck is invoked.

2.2 sigCheckClassifier

The first step is to assess the baseline performance of the identified signature. This is accomplished by a call to sigCheckClassifier, which uses the features identified in the signature on the training samples (all the samples not designated as validation samples) to train a classifier based on the phenotypic categories. In this case, 274 samples are used to train the classifier, utilizing the default machine learning method (a support vector machine). Next, the classifier is used to predict the phenotype of the validation samples (45 samples in this case). The results can be seen as follows:

```
> nkiResults$checkClassifier

$sigPerformance
[1] 0.7333333

$confusion
     predicted
given  0 1
    0 30 2
    1 10 3

$modePerformance
[1] 0.7111111
```

The baseline performance of the signature is the proportion of validation samples that are correctly classified; in this case, that value is 0.733 (33 out of 45). The confusion matrix breaks this down in more detail, showing the numbers of true and false positives as well as true and false negatives.

sigCheckClassifier also computes the performance of a theoretical "mode" classifier. This is the performance that would result if the classifier always made the same prediction, no matter what the data. The prediction is equal to the phenotype category that appears most often in the training set. In the current example, more samples in the training set come from patients who do not experience a metastatic recurrence:

```
> sum(nki$e.dmfs[1:274]==0)

[1] 178

> sum(nki$e.dmfs[1:274]==1)

[1] 96
```

so the mode classifier would always predict 0. In the validation set, an even higher proportion of the samples come from patients who do not experience a relapse:

```
> sum(nki$e.dmfs[274:319]==0)

[1] 33

> sum(nki$e.dmfs[274:319]==1)

[1] 13
```

yielding a mode performance of 0.711.

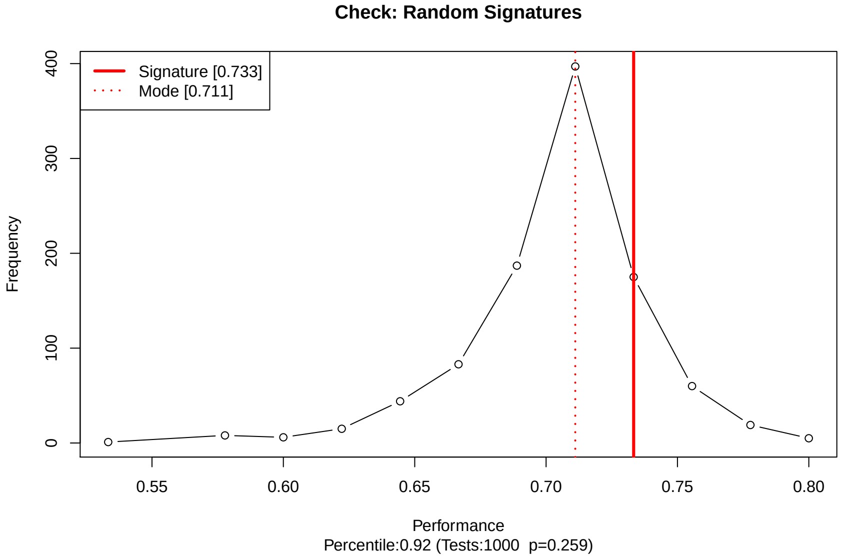

Figure 1: **Results of** `sigCheck` **for NKI Breast Cancer dataset checking v'ant Veer signature against 1,000 random gene signatures.** (Generated using `data(nkiResults)`)

In each of the results plots, the signature classifier performance is indicated by a solid red vertical line, while the mode classifier performance is indicated by a dotted red vertical line. A p-value is also provided, based on the null hypothesis that the signature classifier is no better at predicting phenotype than the classifiers being checked, as follows:

2.3 `sigCheckRandom`

The first check is to compare the performance of the signature against signatures randomly sampled from the dataset. This gives insight into the success of the feature selection process by which the set of features was honed. The number of features in each random signature is equal to the number of features in the identified signature. Figure 1 shows the performance of the v'ant Veer signature on the NKI breast cancer dataset compared to 1,000 randomly selected signatures. The mean performance of the random signatures is equal to that of the mode signature. While the identified signature performance is better than this, some 259 of the random signatures perform as well *or better* than the identified signature. The resulting p-value indicates that the identified signature does not perform significantly better than random signatures.

warning: this should not be taken to suggest that selecting a random signature with better performance on the validation set is a good idea, as it will almost certainly represent an over-fitting to the validation samples. Indeed, performance on validation samples should never *be used in selecting features for a signature.*

2.4 `sigCheckKnown`

The next check is to compare the performance of the signature against known gene signatures that have been used to predict other phenotypes. Figure 2 shows how the performance of the v'ant Veer signature compares to other known cancer-related signatures at classifying the validation samples. While performance overall of the known signatures is skewed higher than for random signatures, the mode performance is equal to the mean performance of the known signatures, and the signature being checked is not among the very best performers. Indeed, some 31% of the known signatures perform as well *or better* than the identified signature. The resulting p-value indicates that the specified signature does not perform significantly better than the set of known cancer signatures as a whole.

The previously identified signatures that outperform the derived signature can be identified as follows:

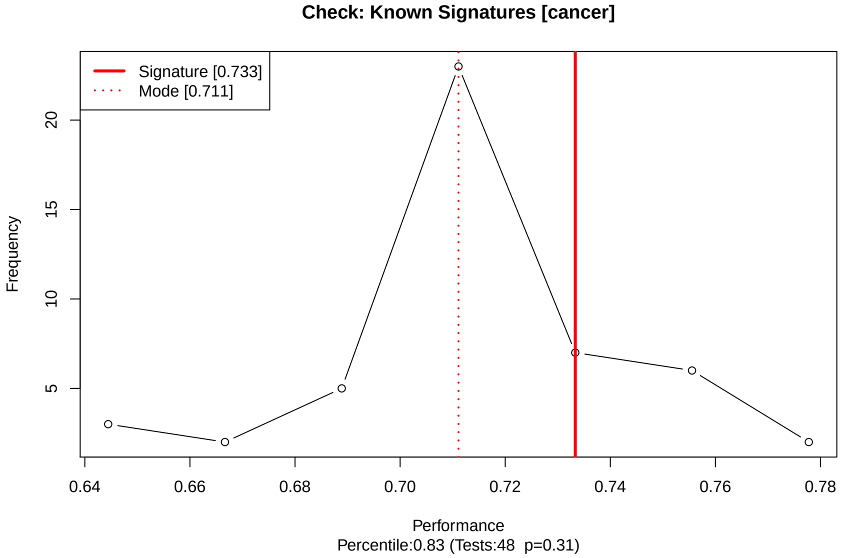

Figure 2: **Results of** `sigCheck` **for NKI Breast Cancer dataset checking v'ant Veer signature against 48 known cancer gene signatures.** (Generated using `data(nkiResults)`)

```
> knownHigh <- nkiResults$checkKnown$performanceKnown >
+              nkiResults$checkClassifier$sigPerformance
> nkiResults$checkKnown$performanceKnown[knownHigh]
```

```
       ABBA        CHANG          HUA    SHIPITSIN     VANTVEER         WELM
  0.7555556    0.7555556    0.7555556    0.7555556    0.7777778    0.7555556
       WEST WONG-MITOCHON
  0.7555556    0.7777778
```

2.5 `sigCheckPermuted`

The final two plots show the results of running `sigCheckPermuted` with two different parameter settings. These represent a sort of reality check by seeing how the gene signature performs on randomly permuted data.

In the first check, the expression values for each feature (specific rows in the expression matrix) are permuted, so the expression values are maintained but assigned randomly to samples. Each feature is permuted independently. Figure 3 shows the results. For most of the datasets permuted in this manner, the performance of the signature is equal to that of the mode classifier, as the best strategy in the face of such noisy data is to always predict the most frequent phenotype. In a small number of cases the performance of the signature is as good or even better on permuted data than on the real data, but this is expected by chance. Only 20 of the permuted datasets enable performance as good or better than using the original data. The empirical p-value indicates that the specified gene signature performs significantly better on the real data than on the permuted data.

For the second check, the phenotype category assignments are permuted across the samples. Figure 4 shows the results. Similar to datasets permuted by features, the performance of the signature equals that of the mode classifier. Likewise,the signature is as good or even better on permuted data than on the real data in some cases, as expected by chance. Only a few (21) enable performance as good or better than using the original data, with a resulting p-value indicating that the specified signature performs significantly better using the correct outcomes rather than randomly assigned ones.

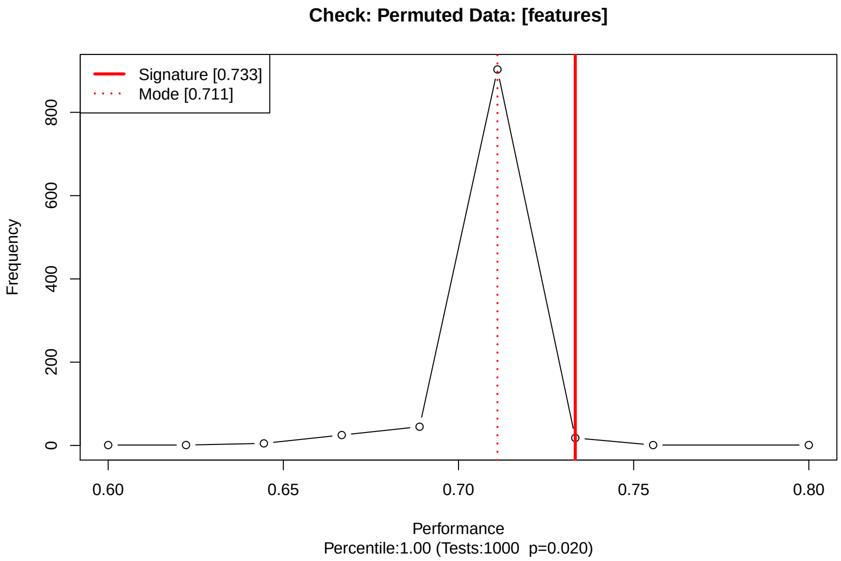

Figure 3: **Results of** `sigCheck` **for NKI Breast Cancer dataset checking v'ant Veer signature using permuted expression data** (Generated using `data(nkiResults)`)

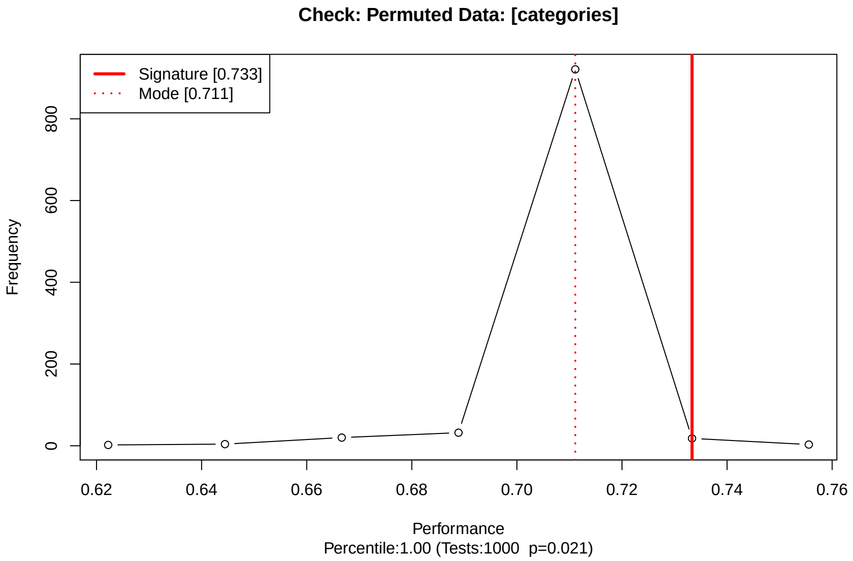

Figure 4: **Results of** `sigCheck` **for NKI Breast Cancer dataset checking v'ant Veer signature using permuted category assignments** (Generated using `data(nkiResults)`)

2.6 Running without a separate validation set using leave-one-out cross-validation on the training set

In the sample so far, we have assumed the presence of a validation set separate from the training samples used to derive the gene signature. While this represents the optimal methodology, when a signature is being developed, there may not yet be a validation set. In this case, a computational strategy known as leave-one-out cross-validation (LOO-XV) will automatically be employed by *SigCheck*. Instead of using all the training samples to construct a single classifier, a separate classifier is trained for each sample, using all the other samples. In the current example, if no validation samples are specified, 319 classifiers will be trained, each being used to predict the phenotype category of a single sample using the remaining 318 samples. When checking against other (random or known) signatures, or permuted datasets, each signature or dataset would require 319 classifiers to be trained. This can be computationally intensive (see note below on use of parallel computation in this case).

For example, the performance of the signature over all of the samples can be computed using LOO-XV:

```
> results <- sigCheckClassifier(nki, classes="e.dmfs",
+                               signature=vantveer,
+                               annotation="HUGO.gene.symbol")
> results

$sigPerformance
[1] 0.7210031

$confusion
     predicted
given   0   1
    0 191  19
    1  70  39

$modePerformance
[1] 0.6583072
```

Note that the mode performance is worse over the entire training set as the numbers of patients in each category are more blanaced.

3 Technical notes

3.1 Use of BiocParallel and parallel

This note shows how to control the parallel processing in *SigCheck*.

There are two different aspects of *SigCheck* that are able to exploit parallel processing. The primary one is when multiple signatures or datasets are being evaluated independently. This include the `nIterations` random signatures in `sigCheckRandom`, the database of known signatures in `sigCheckKnown`, and the `nIterations` permuted datasets in `sigCheckPermuted`. In this case, the *BiocParallel* package is used to carry out these comparisons in parallel. By default, *BiocParallel* uses *parallel* to run in multicore mode, but it can also be configured to schedule a compute task across multiple computers. In the default multicore mode, it will use all of the cores on your system, which can result in a heavy load (especially if there is inadequate memory). You can manually set the number of cores to use as follows:

```
> CoresToUse <- 6
> library(BiocParallel)
> mcp <- MulticoreParam(workers=CoresToUse)
> register(mcp, default=TRUE)
```

which limits the number of cores in use at any one time to six. If you want to use only one core (serial mode), you can set `CoresToUse <- 1`, or `register(SerialParam())`.

The other aspect of processing that can use multiple processor cores is when performing leave-one-out cross-validation (LOO-XV). In this case, the underlying *MLInterfaces* package takes care of the parallelization using the *parallel* package. You can set the number of cores that will be used for this as follows:

```
> options(mc.cores=CoresToUse)
```

Note that in the LOO-XV case, as every random or known signature, or permuted dataset, requires parallel evaluation of cross-validated classifiers, the parallelization at the level of `nIterations` is disabled automatically.

4 Acknowledgements

We would like to acknowledge everyone in the Bioinformatics Core at Cancer Research UK's Cambridge Institute at the University of Cambridge, as well as members of the Ponder group (particularly Kerstin Meyer), for their support and contributions.

5 Session Info

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

- R version 3.1.3 (2015-03-09), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `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`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.28.2, Biobase 2.26.0, BiocGenerics 0.12.1, BiocParallel 1.0.3, GenomeInfoDb 1.2.4, IRanges 2.0.1, MLInterfaces 1.46.0, S4Vectors 0.4.0, SigCheck 1.0.2, XML 3.98-1.1, annotate 1.44.0, breastCancerNKI 1.3.1, cluster 2.0.1, e1071 1.6-4
- Loaded via a namespace (and not attached): BBmisc 1.9, BatchJobs 1.6, BiocStyle 1.4.1, DBI 0.3.1, MASS 7.3-39, RSQLite 1.0.0, base64enc 0.1-2, brew 1.0-6, checkmate 1.5.1, class 7.3-12, codetools 0.2-11, digest 0.6.8, fail 1.2, foreach 1.4.2, gdata 2.13.3, genefilter 1.48.1, gtools 3.4.1, iterators 1.0.7, pls 2.4-3, rda 1.0.2-2, rpart 4.1-9, sendmailR 1.2-1, sfsmisc 1.0-27, splines 3.1.3, stringr 0.6.2, survival 2.38-1, tools 3.1.3, xtable 1.7-4

References

[1] David Venet, Jacques E Dumont, and Vincent Detours. Most random gene expression signatures are significantly associated with breast cancer outcome. *PLoS computational biology*, 7(10):e1002240, 2011.

[2] Laura J van't Veer, Hongyue Dai, Marc J Van De Vijver, Yudong D He, Augustinus AM Hart, Mao Mao, Hans L Peterse, Karin van der Kooy, Matthew J Marton, Anke T Witteveen, et al. Gene expression profiling predicts clinical outcome of breast cancer. *nature*, 415(6871):530–536, 2002.

[3] Marc J Van De Vijver, Yudong D He, Laura J van't Veer, Hongyue Dai, Augustinus AM Hart, Dorien W Voskuil, George J Schreiber, Johannes L Peterse, Chris Roberts, Matthew J Marton, et al. A gene-expression signature as a predictor of survival in breast cancer. *New England Journal of Medicine*, 347(25):1999–2009, 2002.