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

Rory Stark

Edited: December 9, 2015; Compiled: April 30, 2018

Contents

1 Introduction 1

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

3 Example: Survival analysis 5

3.1 Call: `sigCheck` 6

3.2 Call: `sigCheckAll` 7

3.3 Scoring methods for dividing samples into survival groups 9

4 Example: Classification Analysis 11

4.1 Classifiers for scoring survival groups 11

4.2 Checking classifier performance independently of survival 12

4.3 Classification without a validation set: leave-one-out cross-validation 15

5 Technical notes 15

5.1 Obtaining gene signatures from MSigDB 15

5.2 Use of *BiocParallel* and *parallel* 15

6 Acknowledgements 16

7 Session Info 16

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 to distinguish phenotypical classes. 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 dataset: 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, survival data, 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"
 [5] "filename"      "size"          "age"           "er"
 [9] "grade"         "pgr"           "her2"          "brca.mutation"
[13] "e.dmfs"        "t.dmfs"        "node"          "t.rfs"
[17] "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
 [16] 4160 4322 4646 4424 3734 3789 3692 2675 2686 4024 2486 2482 1997 2150 1891
 [31]  398  375 1792 1715  839  410 1691 1787  979  295  748  978  724 1106  785
 [46]  807  777 1347  843   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
 [61]   NA   NA   NA   NA   NA   NA   NA 1201 1809  672  701  929  325 1167  792
 [76]  464  364 4537 3117 1937 1911 3643 3688 5412 5209 2427 2830 2308 1703 3192
 [91] 2764 2665  407 2143 3159 2555 3408 1256 5599 1269 4663 2029  512 5528 5160
[106] 2004 1335  588 6699 6297  351 5118  430 5517  340 6450 2876 1027 1624 5898
[121] 2969 5593 5778 2069 3814  589 5597 5437 4876  507 5022 2774 4592 3260 4812
[136] 4662  955 3951 4134 4332  442 1950 4288 4567 4114 4410 2721 4652  985 4322
[151] 4553  749 4089 4035 3411 3984 3933 4091 1234 4031 3703 3526 3899 2398 4092
[166] 4436  721 2731 3781  627  855 3591 3772 3790  823 3660  972 1226  447  591
[181]   99 1308 5158 2380  907  421  674 2956 1496  732 3646 4217 1526 2059 1802
[196] 1942 2034 3436 3332 1676 3283  839 1869 2015 3033 3139  812 2649 2480 2561
[211] 2531 2589  342   94 2565 2649 2556 2164   20  237 1868 1940    9 2681 2098
[226] 1945 1430 2108 1806 2216  129 1816 4256 3056 2306 2244 2029  119 3505 3454
[241] 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
[271]  788 2919 3103 2810 2731 2706  762 2317 6060 1140  633 5607 2414 2511 2555
[286] 2601 1724 2254 2361 1200 2384 2122 1769 2250 2208 2270 2127 2279 2198 2027
[301] 1953 1921 1816 6604 6387 6265  209 3495 1190 3652  719 2839  979 6363 3115
[316] 5084 5064 4653 2839 4048  711 3000 2639 1249 2485 1282 2254 2036 2085 4095
[331] 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
 [26]  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 NA
 [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA  1  1  1  1  1  1  1  1
 [76]  1  1  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  1  0  1  0  1
[101]  0  0  1  0  0  0  1  1  0  0  1  1  1  0  1  0  0  1  1  0  1  0  0  0  1
[126]  1  0  0  0  1  0  1  0  1  0  0  1  0  0  0  1  0  0  0  0  0  0  0  1  0
```

```
[151]  1  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0  1  1  0  1  1  0  0  0  1
[176]  0  0  1  1  1  1  1  0  0  0  1  1  0  1  1  0  0  0  0  0  0  0  0  1  1
[201]  1  1  1  1  1  0  1  0  0  0  0  0  1  0  0  0  0  0  0  1  0  0  0  0  0
[226]  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
[251]  0  1  0  1  1  0  1  1  1  0  1  0  0  0  0  0  1  0  0  0  1  0  0  0  0
[276]  0  0  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[301]  0  0  1  0  0  0  1  1  1  1  1  0  1  0  1  0  0  0  0  0  1  0  0  1  0
[326]  0  0  0  0  1  0  1  0  1  0  0  0
```

A number of patient samples do not have DMFS data available. Currently, *SigCheck* can not handle NAs in the metadata, 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 three sets of gene signatures, including a set of cancer signatures. [1] The van't Veer signature is one of the signatures in the known cancer gene signature set:

[1] See the Technical Notes section for information on how to obtain more signature lists.

```
> 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"
 [7] "COL4A2"    "CP"        "DCK"       "ECT2"      "EXT1"      "FLT1"
[13] "GNAZ"      "GSTM3"     "IGFBP5"    "MCM6"      "MMP9"      "ALDH6A1"
[19] "OXCT1"     "PEX12"     "PGK1"      "QDPR"      "RAD21"     "RFC4"
[25] "SLC2A3"    "STK3"      "TGFB3"     "BTG2"      "SERF1A"    "CDC42BPA"
[31] "TMEFF1"    "FGF18"     "GMPS"      "WISP1"     "PRC1"      "CCNB2"
[37] "CCNE2"     "MELK"      "NDC80"     "PECI"      "PITRM1"    "NMU"
[43] "GCN1L1"    "ESM1"      "AKAP2"     "ORC6L"     "BBC3"      "UCHL5"
[49] "DTL"       "RAB6B"     "EGLN1"     "C20orf46"  "STK32B"    "DEPDC1"
[55] "C1orf106"  "C16orf61"  "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.

The final aspect of this dataset involve its division into a training (or discovery) set of samples and a validation set of samples. The training set should include all the samples that were used in deriving the gene signature. It is expected that the signature should perform optimally on these samples. A validation n set of samples that played no role in deriving the signature is required to test the efficacy of the signature. For the NKI dataset, the `varLabel` named `"series"` specifies which samples were int he original trainingset, and which were profiled in the follow-on experiment: For this example, we will treat the first 100 samples as the training set.

```
> table(nki$series)
```

```
 NKI NKI2
  99  220
```

3 Example: Survival analysis

In this section, we will use the survival data to test the ability of the van't veer signature to predict outcome in the validation set. The most straightforward way to accomplish this involved two function calls. The first is a call to `sigCheck`, which sets up the experiment, establishes the baseline performance, generates Kaplan-Meier plots, and returns a newly constructed object of class `SigCheckObject`. The second call, to `sigCheckAll`, runs a default series of tests before plotting and returning the results.

3.1 Call: `sigCheck`

The `sigCheck` function is the constructor for `SigCheckObject`s. It requires a number of parameters that are used to define the experimental data and signature.

The first required parameter is `expressionSet`, an `ExpressionSet` object that contains all the experimental data. In this example, `expressionSet=nki`. The next required parameter is `classes`, a character string indicating which of `varLabels(expressionSet)` contains the binary class data. In this example, `classes=e.dmfs`.

For a survival analysis, the `survival` parameter indicates which of `varLabels(expressionSet)` contains the numeric survival data. In this example, `classes=t.dmfs`

The `signature` parameter contains the signature that will be checked. This is most easily specified as a vector of feature names (ie gene IDs) that match features in the `annotation` parameter. The `annotation` parameter specifies which of `fvarLabels(expressionSet)` contains the feature labels. In this example, `signature=knownSignatures$cancer$VANTVEER`, and `annotation="HUGO.gene.symbol"`.

If the data are divided into training and validation samples, the samples that comprise the validation set should be specified using the `validationSamples` parameter.

The remaining parameters are optional, with usable defaults, and will be discussed in subsequent sections.

Putting this all together, the call to set up the experiment is:

```
> check <- sigCheck(nki, classes="e.dmfs", survival="t.dmfs",
+                   signature=knownSignatures$cancer$VANTVEER,
+                   annotation="HUGO.gene.symbol",
+                   validationSamples=which(nki$series=="NKI2"))
> check

ExpressionSet (storageMode: lockedEnvironment)
assayData: 24481 features, 319 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: NKI_4 NKI_6 ... NKI_404 (319 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: HUGO.gene.symbol
```

By default, this will generate two Kaplan-Meier plots, as shown in Figure 1. The first shows the performance on the training set, and the other shows performance on the validation set. The method that uses the signature to separate the samples into two groups, based on taking the first principal component (as described in [1]), is such that the High/Low groups can be flipped in the two charts without concern.

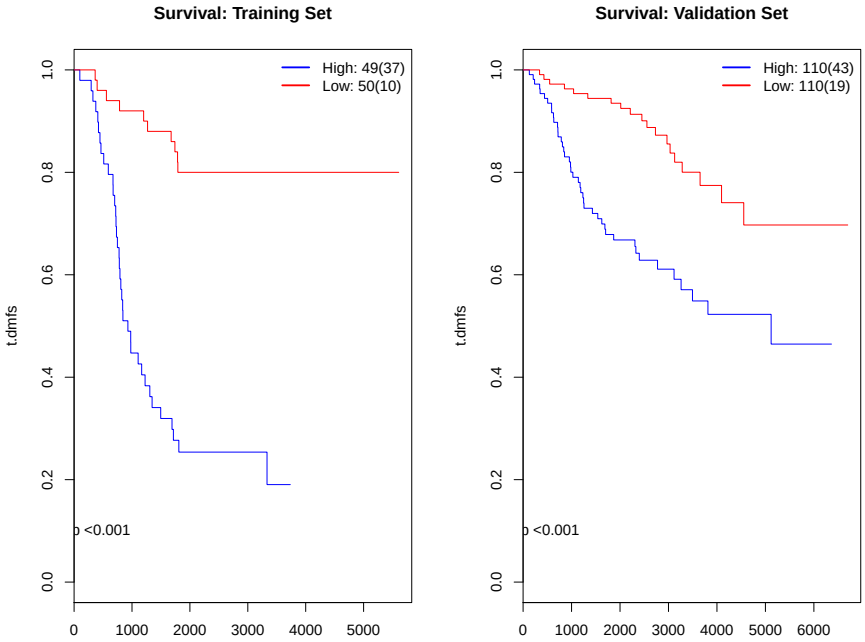

Figure 1: Results of `sigCheck` for NKI Breast Cancer dataset checking v'ant Veer signature

These plots show very strong performance on the training set, with distinct (but reduced) separation on the validation set. Both p-values are reported as $p<0.001$. Examination of the resulting object shows the actual p-value computed for the validation set:

```
> check@survivalPval

[1] 3.984627e-05
```

3.2 Call: `sigCheckAll`

From this analysis, it appears that the signature does indeed have power in distinguishing between two distinct survival groups. The next question is how unique these specific genes in the signature are for this task. the function `sigCheckAll` will run a series of checks for comparison purposes. In each check, some background performance distribution is computed, and the performance of the signature is compared by calculating an empirical p-value.

The four tests include a distribution of p-values computed using randomly selected signature of the same size (number of features). This check can be run separately using `sigCheckRandom`. The second test compares the performance of this signature to a set of previously identified ones. This check can be run separately using `sigCheckKnown`. The third and fourth tests compare performance of the signature on the dataset to permuted versions of the dataset; specifically permuting the `class survival` metadata, as well as permuting the feature data by randomly re-assigning the expression values for each feature across the samples. The permuted data/metadata check can be run separately using `sigCheckPermuted`.

The parameters to `sigCheckAll` include the `SigCheckObject` constructed by the previous call to `sigCheck`. The `iterations` parameter determines the size of the null distribution for the random and permuted checks. The `known` parameter specifies a set of known signatures to use for the second check, with the default being the 48 cancer signatures identified by [1].

The number assigned to `iterations` will determine the accuracy of the p-value calculated for the random and permuted tests. A value of at least 1000 is preferred to get a meaningful value:

```
> nkiResults <- sigCheckAll(check, iterations=1000)
```

As this vignette will take too long to run automatically with `iterations=1000`, the results have been pre-computed and included as a data object with the (SigCheck) package:

```
> data(nkiResults)
```

By default, `sigCheckAll` will generate plots of the results. These can be re-generated using `sigCheckPlot`:

```
> sigCheckPlot(nkiResults)
```

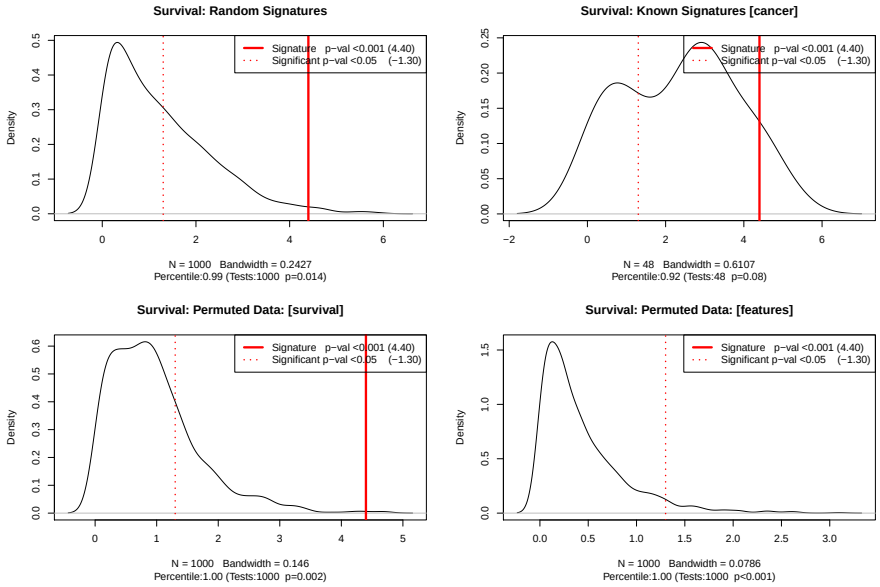

Figure 2: Results of `sigCheckPlot` for NKI Breast Cancer dataset checking v'ant Veer signature

In each of the four plots, the background distribution is plotted, with the x-axis representing survival p-values on a -log 10 scale, and the y-axis representing how many of the background tests produced p-values at that level. The solid red vertical line shows the performance of the signature being tested. The further to the right of the distribution this line is, the more uniquely it performs for this check. In some cases, if the performance of the signature is out fo the range of the background distribution, this line will not be drawn. The vertical red dotted line shows where a "significant" result (p=0.05) would lie relative to the background distribution.

The upper-left plot shows performance of the v'ant veer signature relative to 1,000 randomly chosen signature of comprised of an equal number of gene. The upper-right plot shows the performance compared to 48 known cancer prognostic signatures, while the two lower plots show performance on permuted data (survival and gene expression across each feature).

The result returned by the call to `sigCheckAll` is a list of results for each test:

```
> names(nkiResults)

[1] "checkRandom"           "checkKnown"            "checkPermutedSurvival"
[4] "checkPermutedFeatures"
```

The adjusted p-value computed for each check can be retrieved as follows:

```
> nkiResults$checkRandom$checkPval

[1] 0.014

> nkiResults$checkKnown$checkPval

[1] 0.083

> nkiResults$checkPermutedSurvival$checkPval

[1] 0.002

> nkiResults$checkPermutedFeatures$checkPval

[1] 0
```

3.3 Scoring methods for dividing samples into survival groups

The survival analysis depends on a method for using the signature to divide the samples into survival groups before computing the p-value for how their survival times differ. *SigCheck* divides this task into two steps. The first is to calculate a score for each sample based on the feature values for each feature in the signature. The second step is to divide the samples into groups based on these scores.

[1] discuss this issue, and recommend first determining the score for each sample by computing the first principal component of the expression matrix (using only the features in the signature). They then suggest that the samples can be divided into two groups based on whether their score is above or below the median score. This is the default method used in *SigCheck*.

There are options for both the scoring and dividing steps. The scoring method is determined by the value of the parameter `scoreMethod` for the function `sigCheck`. The default value is `"PCA1"`. `scoreMethod` may also be `"High"`, which simply computes the mean expression value for each sample across the signature. This enables the samples to be divided into a "High" expression group and a "Low" expression group. Another option is to use a machine-learning classifier to do the scoring (`scoreMethod="classifier"`); the next section discusses how to use classifiers in *SigCheck*.

Finally, `scoreMethod` can be set to a user-defined function if you want to do your own mapping of expression value to scores. The user-defined scoring function should take an `ExpressionSet` as a parameter. This `ExpressionSet` will have as many rows as features that match the signature, and as many columns as there are samples (either training or validation samples). It should return a vector of scores, one for each sample.

The second step is to use the scores to divide the samples into groups. The `threshold` parameter controls how this is accomplished. Currently, the package only support division into either two or three groups, determined by which samples have scores below the specified percentiles, and which are greater than or equal to the specified percentile. The default is to use the median as the threshold, which will split the samples into two groups of approximately equal size. In many cases, however, the real power of a signature is not to split the samples into two equal sized groups, but rather to identify a subset of samples that have a distinct outcome. Setting `threshold=.66`, for example, will split the samples into a larger group with two-thirds of the sample, and a smaller group containing the one-third of samples with the highest score. If you are using a score such as `scoreMethod="High"`, these would be the samples with the highest mean expression over the signature, while setting `scoreMethod=.33` would split off a smaller group of sample with the lowest mean expression. By specifying two percentile cutoffs, you can split the samples into three groups: one with high scores, one with low scores, and one with intermediate scores. The performance of the signature will computed using only the samples in the high and low groups.

To see how all this works, consider some alternative scoring and grouping methods for the sample data set. For example, we can see how the v'ant Veer signature performs when looking at overall expression levels. The code below takes advantage of the ability to create a new `SigCheckObject` from an existing one:

```
> par(mfrow=c(2,2))
> p5 <- sigCheck(check,
+               scoreMethod="High",threshold=.5,
+               plotTrainingKM=F)@survivalPval
> p66 <- sigCheck(check,
+               scoreMethod="High",threshold=.66,
+               plotTrainingKM=F)@survivalPval
> p33 <- sigCheck(check,
+               scoreMethod="High",threshold=.33,
+               plotTrainingKM=F)@survivalPval
> p33.66 <-sigCheck(check,
+               scoreMethod="High",threshold=c(.33,.66),
+               plotTrainingKM=F)@survivalPval
> p5

[1] 0.0003488032

> p66

[1] 0.003568654

> p33

[1] 1.72021e-05

> p33.66

[1] 1.002417e-05
```

Figure 3 shows the baseline performance of the signature using different thresholds. The one that splits off high and low expression groups, and eliminates the middle samples, performs the best. However to know how unique this performance is, the random and known checks would have to be repeated using exactly the same evaluation criteria as for the main signature.

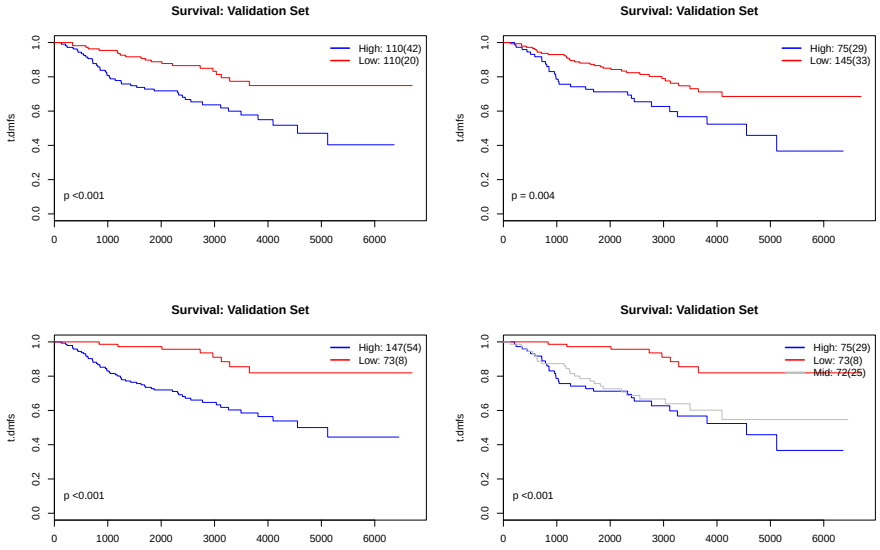

Figure 3: Results of `sigCheckPlot` for NKI Breast Cancer dataset checking v'ant Veer signature, using "High" scoring method and different percentile cutoffs

4 Example: Classification Analysis

SigCheck uses the *MLInterfaces* package to enable a wide range of machine-learning algorithms to be applied to expression data. Classifiers constructed in this way use only the binary `classes` metadata, and not the survival data, to predict what class each sample belongs to.

Classifiers can be applied in two distinct ways: as a scoring method for determining survival groups, and for assessing the classification potential of signatures when survival data is unavailable.

4.1 Classifiers for scoring survival groups

As many classifiers generate a score for each sample (representing, for example, the probability that a sample belongs to a specific class), they can be used as the basis for dividing samples into survival groups that can be uses to assess survival analysis performance. This is accomplished by setting `scoreMethod="classifier")` when invoking `(`sigCheck).

When specifying a classifier, the `classifierMethod` parameter is used to determine what type of classifier is to be used. This can be any classifier supported by the *MLInterfaces* package. The default, `svmI`, uses a Support Vector Machine. When invoked, the training set samples will be used to construct a classifier that distinguished between the two classes specified in the `classes` parameter. When the training set is divided into survival groups, the resulting classifier is used to generate scores for all the samples. These scores are then subjected to the `threshold` parameter as with the other `scoreMethod`s.

To see this in action with the sample dataset, all that is required is to set `scoreMethod="classifier"`:

```
> check <- sigCheck(check, scoreMethod="classifier")

Called from: .method@converter(ans, data, trainInd)
debug: teData = data[-trainInd, ]
debug: trData = data[trainInd, ]
debug: tepr = predict(obj, teData, decision.values = TRUE, probability = TRUE)
debug: trpr = predict(obj, trData, decision.values = TRUE, probability = TRUE)
debug: new("classifierOutput", testPredictions = factor(tepr[seq_len(length(tepr))]),
    trainPredictions = factor(trpr[seq_len(length(trpr))]), testScores = attr(tepr,
        "probabilities"), trainScores = attr(trpr, "probabilities"),
    RObject = obj)

> check@survivalPval

[1] 0.000122871
```

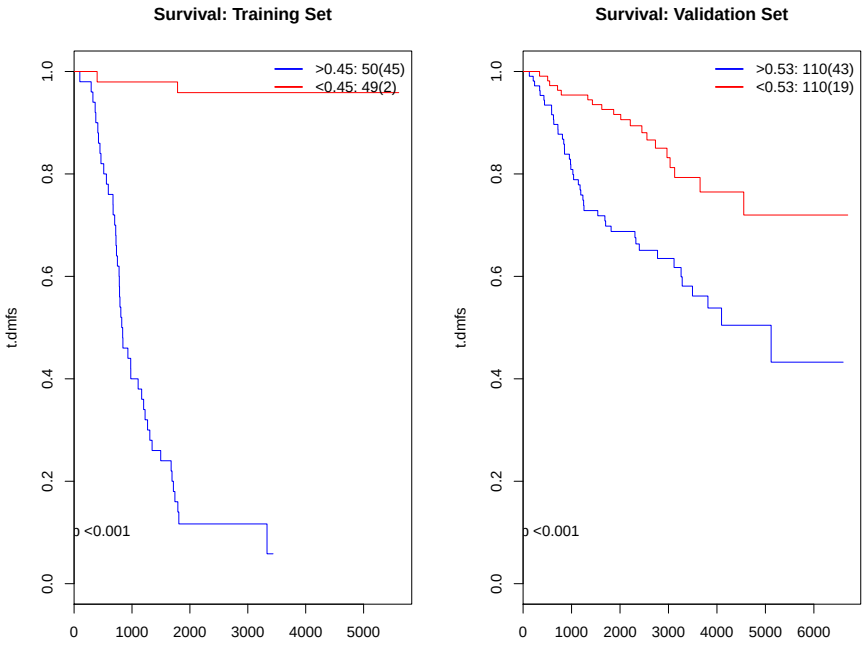

Figure 4: Results of `sigCheck` for NKI Breast Cancer dataset checking v'ant Veer signature, using a Support Vector Machine to classify samples into survival groups

4.2 Checking classifier performance independently of survival

Another way of evaluating gene signatures is on how well they perform on the classification task itself. For example, if the goals was to generate a gene signature to predict susceptibility to a disease, the distinction between those who developed the disease and those who remain disease-free may be important than the associated timeframes. In some cases, only class data (recurrence/non recurrence; death/survival) may be available for the samples, and no real-valued timescales.

In these cases (as well as cases where survival data are available but classification is of interest), the classification abilities of a signature can be checked in similar manner to survival.

To generate such an analysis, simply leave the `survival` unspecified:

```
> check <- sigCheck(nki, classes="e.dmfs",
+                   signature=knownSignatures$cancer$VANTVEER,
+                   annotation="HUGO.gene.symbol",
+                   validationSamples=which(nki$series=="NKI2"),
+                   scoreMethod="classifier")

Called from: .method@converter(ans, data, trainInd)
debug: teData = data[-trainInd, ]
debug: trData = data[trainInd, ]
debug: tepr = predict(obj, teData, decision.values = TRUE, probability = TRUE)
debug: trpr = predict(obj, trData, decision.values = TRUE, probability = TRUE)
debug: new("classifierOutput", testPredictions = factor(tepr[seq_len(length(tepr))]),
    trainPredictions = factor(trpr[seq_len(length(trpr))]), testScores = attr(tepr,
        "probabilities"), trainScores = attr(trpr, "probabilities"),
    RObject = obj)

> check

     predicted
given  0  1
    0 95 63
    1 20 42
ExpressionSet (storageMode: lockedEnvironment)
assayData: 24481 features, 319 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: NKI_4 NKI_6 ... NKI_404 (319 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: HUGO.gene.symbol
```

In this case, no Kaplan-Meier plots are generated as there is no survival data. The baseline classification performance can be examined:

```
> check@sigPerformance

[1] 0.6227273

> check@modePerformance

[1] 0.7181818

> check@confusion

     predicted
```

```
given  0  1
    0 95 63
    1 20 42
```

The first value is the percentage of validation samples correctly classified. The second value is the percentage that would be correctly classified if the classifier just guess the most frequently observed class in the training set (the mode value). The third value is the confusion matrix, which shows how validation samples in each category were classified. This can be interpreted in terms of True Negatives and False Positives (first row), and False Negatives and True Positives (second row).

The classification performance of the signature can be checked against the performance of a background distribution of random or known signatures:

```
> classifyRandom <- sigCheckRandom(check, iterations=1000)
> classifyKnown <- sigCheckKnown(check)
```

As with the previous checks, the results have been pre-computed to save computation time:

```
> data(classifyResults)
```

The results can be plotted:

```
> par(mfrow=c(1,2))
> sigCheckPlot(classifyRandom, classifier=TRUE)
> sigCheckPlot(classifyKnown,  classifier=TRUE)
```

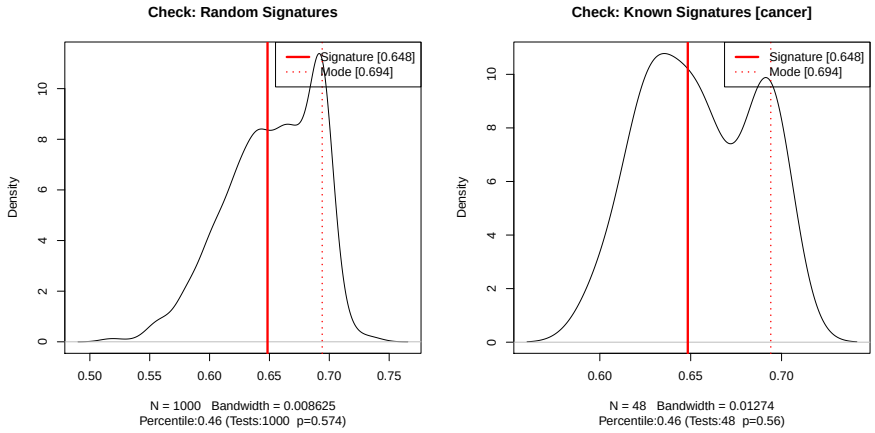

Figure 5: Results of classifier performance check against random and known signature for NKI Breast Cancer dataset checking v'ant Veer signature, using "High" scoring method and different percentile cutoffs

Figure 5 shows the results of a classification analysis. The background distribution based on the classification performance of the random or known signatures, so they are the percentage of validation samples correctly classified. That means that for these plots, better performing signatures are toward the right on the x-axis. Also, the dotted red vertical line represents the performance of a mode classifier. It can be very useful to see what a classifier that always guesses the category more prevalent in the training set for comparison to a curated gene signature and a sophisticated machine learning algorithm.

4.3 Classification without a validation set: leave-one-out cross-validation

coming soon...

5 Technical notes

5.1 Obtaining gene signatures from MSigDB

When checking a signature's performance against known signatures (ie using the `sigCheckKnown` function), the gene signatures available at the Broad Institute's *MSigDB* site (part of *GSEQ* [4]). These are located at the following URL: http://software.broadinstitute.org/gsea/downloads.jsp. Note that you must first register and accept the licence terms, which is why the signatures are not distributed with this package.

The gene sets can be downloaded in `.gmt` format. This can be read in using the `read.gmt` function from the *qusage* package. For example, if you download the oncogenic signatures file, you can retrieve the signatures as follows:

```
> c6.oncogenic <- read.gmt('c6.all.v5.0.symbols.gmt')
> check.c6 <- sigCheckKnown(check, c6.oncogenic)
> sigCheckPlot(check.c6)
```

5.2 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 `iterations` random signatures in `sigCheckRandom`, the database of known signatures in `sigCheckKnown`, and the `iterations` 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 `iterations` is disabled automatically.

6 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.

7 Session Info

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

- R version 3.5.0 (2018-04-23), `x86_64-pc-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`
- Running under: `Ubuntu 16.04.4 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.42.0, Biobase 2.40.0, BiocGenerics 0.26.0, BiocParallel 1.14.0, IRanges 2.14.0, MLInterfaces 1.60.0, S4Vectors 0.18.0, SigCheck 2.12.0, XML 3.98-1.11, annotate 1.58.0, breastCancerNKI 1.17.0, cluster 2.0.7-1, e1071 1.6-8, survival 2.42-3
- Loaded via a namespace (and not attached): BiocStyle 2.8.0, DBI 0.8, DEoptimR 1.0-8, MASS 7.3-50, Matrix 1.2-14, R6 2.2.2, RColorBrewer 1.1-2, RCurl 1.95-4.10, RSQLite 2.1.0, Rcpp 0.12.16, assertthat 0.2.0, backports 1.1.2, base64enc 0.1-3, bindr 0.1.1, bindrcpp 0.2.2, bit 1.1-12, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, class 7.3-14, compiler 3.5.0, crosstalk 1.0.0, digest 0.6.15, diptest 0.75-7, dplyr 0.7.4, evaluate 0.10.1, flexmix 2.3-14, fpc 2.1-11, gbm 2.1.3, gdata 2.18.0, genefilter 1.62.0, ggvis 0.4.3, glue 1.2.0, grid 3.5.0, gtools 3.5.0, htmltools 0.3.6, htmlwidgets 1.2, httpuv 1.4.1, hwriter 1.3.2, igraph 1.2.1, kernlab 0.9-26, knitr 1.20, later 0.7.1, lattice 0.20-35, magrittr 1.5, mclust 5.4, memoise 1.1.0, mime 0.5, mlbench 2.1-1, modeltools 0.2-21, mvtnorm 1.0-7, nnet 7.3-12, pillar 1.2.2, pkgconfig 2.0.1, pls 2.6-0, prabclus 2.2-6, promises 1.0.1, rda 1.0.2-2, rlang 0.2.0,

rmarkdown 1.9, robustbase 0.93-0, rpart 4.1-13, rprojroot 1.3-2, sfsmisc 1.1-2, shiny 1.0.5, splines 3.5.0, stringi 1.1.7, stringr 1.3.0, threejs 0.3.1, tibble 1.4.2, tools 3.5.0, trimcluster 0.1-2, xtable 1.8-2, yaml 2.1.18

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.

[4] Arthur Liberzon, Aravind Subramanian, Reid Pinchback, Helga Thorvaldsdóttir, Pablo Tamayo, and Jill P Mesirov. Molecular signatures database (msigdb) 3.0. *Bioinformatics*, 27(12):1739–1740, 2011.