Computing and plotting agreement among ranked vectors of features with matchBox.

Anuj Gupta[1] and Luigi Marchionni[1]

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

Modified: September 5, 2012. Compiled: January 25, 2026

Contents

1 Introduction 2

2 Preparing the Data 3

2.1 Data structure . . . 3

2.2 Removing redundancy . . . 4

2.3 Merging the data . . . 6

3 Correspondance at the Top Curves 6

3.1 Computing CAT curves without a reference ranking . . . 7

3.2 Computing CAT curves using a reference ranking . . . 7

3.3 Computing Probability Intervals . . . 8

4 Plotting the results 10

4.1 CAT curves based on equal ranks using a reference. . . . 11

4.2 CAT curves based on equal statistics using a reference. . . . 12

4.3 CAT curves based on equal ranks without using a reference. . . . 13

4.4 CAT curves based on equal ranks, unspecified expected proportion of corresponding features. . . . 14

5 System Information 15

6 Literature Cited 15

1 Introduction

The `matchBox` package allows to compare ranked vectors (*i.e.* by differential expression) of features (*i.e.* genes, or probe sets), using Correspondence-At-the-Top (CAT) curves. A CAT curve displays the overlapping proportion between two ranked vectors of identifiers against the number of considered features. This techiques was originally envisaged for comparing differential gene expression results obtained from distinct microarray platforms in different laboratories (see Irizarry et al, Nat Methods (2005) [1]), and further used to compare differential gene expression across distinct studies (see Ross et al, Prostate (2011) [2], and Benassi et al., Cancer Discovery (2012) [3]).

The `matchBox` package contains several utilities enabling the following:

1. To filter redundant features in a `data.frame` (*i.e.* filtering multiple probe sets pointing to the same gene);
2. To merge multiple data sets together based on a common set of identifiers;
3. To compute the CAT curves probability intervals;
4. To nicely plot the CAT curves.

Briefly, CAT curves enable to compare the correspondence of two vectors ordered by a predefined ranking statistics at their top. This is accomplished through:

1. Ordering the two vectors according to a desired statistics (*i.e.* differential gene expression, significance, probability, . . .);
2. Computing the proportion of top ranking elements in common between the two vector for a given vector size (*i.e.* top 5 features);
3. Reiterating the two steps above increasing the vector size up to all common elements (*i.e.* the top 6 identifiers, then the top 7, 8, 9, . . . , all features);
4. Plotting the proportion of common elements against the increasing vector size.

Since CAT curves evaluate agreement at the top of ranked vectors they are particularly useful in gene expression analysis, where only a small fraction of genes is expected to be differentially expressed over the large total number of analyzed genes.

2 Preparing the Data

2.1 Data structure

Download and install the package `matchBox` from Bioconductor.

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

Load the library.

```
> require(matchBox)
```

Load the example data contained in the `matchBox` package.

```
> data(matchBoxExpression)
```

The object `matchBoxExpression` is a list of data.frames containing differential gene expression analysis resulst from three distinct experiments. These data.frames will be used to retrieve the identifiers and the ranking statistics for computing overlapping proportions displayed by the CAT curves. Below is shown the structure of `matchBoxExpression`:

```
> sapply(matchBoxExpression, class)

    dataSetA     dataSetB     dataSetC
"data.frame" "data.frame" "data.frame"

> sapply(matchBoxExpression, dim)

     dataSetA dataSetB dataSetC
[1,]     1375     2358     1800
[2,]        9        9        9

> str(matchBoxExpression)

List of 3
 $ dataSetA:'data.frame':        1375 obs. of  9 variables:
  ..$ SYMBOL   : chr [1:1375] "KLK11" "CDKN1B" "LPAR1" "JAKMIP2" ...
  ..$ GENENAME : chr [1:1375] "kallikrein-related peptidase 11" "cyclin-dependent kinase
  ..$ ENTREZID : chr [1:1375] "11012" "1027" "1902" "9832" ...
  ..$ logFC    : num [1:1375] -1.99 0.974 -1.027 1.376 1.376 ...
  ..$ AveExpr  : num [1:1375] 10.1 10.6 10.7 11.4 11.4 ...
  ..$ t        : num [1:1375] -8.13 6.96 -6.73 6.42 6.42 ...
  ..$ P.Value  : num [1:1375] 8.44e-11 6.02e-09 1.37e-08 4.32e-08 4.32e-08 ...
  ..$ adj.P.Val: num [1:1375] 1.36e-07 4.45e-06 7.02e-06 1.67e-05 1.67e-05 ...
  ..$ B        : num [1:1375] 15.53 11.71 10.96 9.92 9.92 ...
 $ dataSetB:'data.frame':        2358 obs. of  9 variables:
  ..$ SYMBOL   : chr [1:2358] "KLK11" "CDKN1B" "LPAR1" "JAKMIP2" ...
  ..$ GENENAME : chr [1:2358] "kallikrein-related peptidase 11" "cyclin-dependent kinase
```

```
  ..$ ENTREZID : chr [1:2358] "11012" "1027" "1902" "9832" ...
  ..$ logFC    : num [1:2358] 0.885 1.532 0.599 0.714 0.408 ...
  ..$ AveExpr  : num [1:2358] 8.72 8.49 12.29 7.37 10.79 ...
  ..$ t        : num [1:2358] 13.5 12.08 12.07 9.76 9.36 ...
  ..$ P.Value  : num [1:2358] 1.88e-19 2.18e-17 2.23e-17 8.62e-14 3.81e-13 ...
  ..$ adj.P.Val: num [1:2358] 6.96e-16 4.06e-14 4.06e-14 6.12e-11 2.31e-10 ...
  ..$ B        : num [1:2358] 33.7 29.5 28.6 22.2 20.8 ...
 $ dataSetC:'data.frame':        1800 obs. of  9 variables:
  ..$ SYMBOL   : chr [1:1800] "CD38" "C15orf61" "PECI" "PSMC2" ...
  ..$ GENENAME : chr [1:1800] "CD38 molecule" "chromosome 15 open reading frame 61" "perc
  ..$ ENTREZID : chr [1:1800] "952" "145853" "10455" "5701" ...
  ..$ logFC    : num [1:1800] -0.699 0.24 0.366 0.405 0.442 ...
  ..$ AveExpr  : num [1:1800] 12.2 10.8 11.5 12.1 12.6 ...
  ..$ t        : num [1:1800] -10.77 10.11 9.13 9.09 8.82 ...
  ..$ P.Value  : num [1:1800] 1.14e-13 7.96e-13 1.59e-11 1.79e-11 4.11e-11 ...
  ..$ adj.P.Val: num [1:1800] 2.29e-10 1.05e-09 1.22e-08 1.33e-08 2.28e-08 ...
  ..$ B        : num [1:1800] 22.1 20.4 17.6 17.5 16.7 ...
```

2.2 Removing redundancy

In order to compute the CAT curves, and then compare the results obtained from the three experiments, it is necessary to identify the set of **non-redundant** features in common among the three data sets. The `filterRedundant` function enables to remove the redundant features within each `data.frame` prior computing the subset of of common identifiers across the three data sets. The argument `idCol` specifies the index or the name of the column containing redundant identifiers, while the other arguments enable to control the method used to remove the redundancy, as explained in the help for this function.

By default the `filterRedundant` function will keep the feature with the largest absolute ranking statistics, as defined by the `byCol` argument. The `method` argument enables to control which feature to keep and which to discard. Currently available methods are: `maxORmin`, `geoMean`, `random`, `mean`, and `median` (see help for `filterRedundant`).

In the example below we are using an `lapply` call to remove the redundant features from all the three data.frames at once. To this end we will use gene symbols to identify the redundant features and the t-statistics to select which feature to keep.

```
> sapply(matchBoxExpression, function(x) any(duplicated(x[, 1])) )

dataSetA dataSetB dataSetC
    TRUE     TRUE     TRUE

> allDataBySymbolAndT <- lapply(matchBoxExpression, filterRedundant,
                                idCol="SYMBOL", byCol="t", absolute=TRUE)
```

Each data.frame now contains only unique features.

```
> sapply(allDataBySymbolAndT, dim)
     dataSetA dataSetB dataSetC
[1,]     1075     2058     1500
[2,]        9        9        9
> sapply(allDataBySymbolAndT, function(x) any(duplicated(x[, 1])) )
dataSetA dataSetB dataSetC
   FALSE    FALSE    FALSE
```

In the example below we are removing the redundant features using Enrez Gene identifiers to select the redundant features and the **signed** log2 fold-change to select which feature to keep.

```
> sapply(matchBoxExpression, function(x) any(duplicated(x[, 1])) )
dataSetA dataSetB dataSetC
    TRUE     TRUE     TRUE
> allDataByEGIDAndLogFC <- lapply(matchBoxExpression, filterRedundant,
                                  idCol="ENTREZID", byCol="logFC", absolute=FALSE)
```

Each of the data.frame now contains only unique features.

```
> sapply(allDataByEGIDAndLogFC, dim)
     dataSetA dataSetB dataSetC
[1,]     1075     2058     1500
[2,]        9        9        9
> sapply(allDataByEGIDAndLogFC, function(x) any(duplicated(x[, 1])) )
dataSetA dataSetB dataSetC
   FALSE    FALSE    FALSE
```

In the example below we are removing the redundant features using Enrez Gene identifiers to select the redundant features, and the median adjusted P-value will be used to summarize the redundant features.

```
> sapply(matchBoxExpression, function(x) any(duplicated(x[, 1])) )
dataSetA dataSetB dataSetC
    TRUE     TRUE     TRUE
> allDataByEGIDAndMedianFDR <- lapply(matchBoxExpression, filterRedundant,
                                      idCol="ENTREZID", byCol="adj.P.Val", absolute=FALSE,
                                      method="median")
```

Each of the data.frame now contains only unique features.

```
> sapply(allDataByEGIDAndMedianFDR, dim)
     dataSetA dataSetB dataSetC
[1,]     1075     2058     1500
[2,]        9        9        9
> sapply(allDataByEGIDAndMedianFDR, function(x) any(duplicated(x[, 1])) )
dataSetA dataSetB dataSetC
   FALSE    FALSE    FALSE
```

2.3 Merging the data

After removing the redundant features it is now possible to obtain the common unique features across all the data sets using the function `mergeData`. This function finds the intersection across all the data.frames, and extract the desired ranking statistics from each one.

```
> data <- mergeData(allDataBySymbolAndT, idCol="SYMBOL", byCol="t")
```

The object `data` is a `data.frame` containing only the common set of features across all three data.frames and the ranking statistics values of choice collected from each data.frame.

```
> sapply(allDataBySymbolAndT, dim)

     dataSetA dataSetB dataSetC
[1,]     1075     2058     1500
[2,]        9        9        9

> dim(data)

[1] 506   4

> str(data)

'data.frame':        506 obs. of  4 variables:
 $ commonID  : chr  "A1CF" "AARS2" "ABCB1" "ABCG8" ...
 $ dataSetA.t: num  2.82 1.91 -2.27 2.26 -1.93 ...
 $ dataSetB.t: num  -3.61 -2.8 -3.07 3.07 -2.81 ...
 $ dataSetC.t: num  -3.454 -0.319 1.324 -1.797 -4.777 ...
```

3 Correspondance at the Top Curves

The `computeCat` R function enables to compute the overlapping proportions of features among ranked vectors of identifiers. Such proportions will be subsequently used to produce the CAT curves. When computing CAT curves a number of paramenters must be specificied to control the behaviour of the `computeCat` function. In particular the following information will determine how the vectors of will be ordered, and which vectors will be compaired, and therefore must be carefully considered:

- Whether or not the ranking statistics used to order the features is signed (*i.e.* t-statistics or F-statistics like);
- Whether the ranking statistics should be used to order the features by decreasing or increasing order. For instance in the case of differential gene expression between group A and B:
  - Ordering by **decreasing signed** t-statistics will compute the CAT curve for the genes up-regulated in group A compared to group B;

- Ordering by **increasing signed** t-statistics will compute the CAT curve for the genes down-regulated in group A compared to group B;
- Ordering by **decreasing absolute** t-statistics will compute the CAT curve for the differentiallly expressed genes between the two groups;
- Ordering by **increasing absolute** t-statistics will compute the CAT curve for the genes that **are not differentially expressed** between the two groups;

- Whether to compare all possible vector combinations, or to select one of the vectors as the reference;
- Whether the overlapping proportion should be computed using equal ranks or equal statistics;

3.1 Computing CAT curves without a reference ranking

The example below shows how to compute CAT curves **without** selecting one of the vectors as the reference, using **decreasing** ranks (*i.e.* up-regulated genes are at the top of the list):

```
> catHigh2LowNoRefByEqualRanks <- computeCat(data = data, idCol = 1,
                                              method="equalRank", decreasing=TRUE)
```

The example below shows to compute CAT curves **without** selecting a vector as the reference, using **increasing** ranks (*i.e.* down-regulated genes are at the top of the list):

```
> catLow2HighNoRefByEqualRanks <- computeCat(data = data, idCol = 1,
                                              method="equalRank", decreasing=FALSE)
```

3.2 Computing CAT curves using a reference ranking

The example below shows how to compute CAT curves selecting one of the vectors as the reference, using decreasing ranks:

```
> catHigh2LowWithRefByEqualRanks <- computeCat(data = data, idCol = 1,
                                                ref="dataSetA.t",  method="equalRank", dec
```

The `catHigh2LowWithRefByEqualRanks` contains the computed overlaps, for decreasing ranks (*i.e.* up-regulated genes are at the top of the list):

```
> str(catHigh2LowWithRefByEqualRanks)

List of 2
 $ dataSetA.t.vs.dataSetB.t:List of 1
  ..$ cat: num [1:506] 0 0.5 0.333 0.5 0.6 ...
 $ dataSetA.t.vs.dataSetC.t:List of 1
  ..$ cat: num [1:506] 0 0 0 0 0 0 0 0 0 0 ...
```

The example below shows to compute CAT curves selecting one of the data set as reference, using decreasing t-statistics:

```
> catHigh2LowWithRefByEqualStats <- computeCat(data = data, idCol = 1, ref="dataSetA.t",
                                                method="equalStat", decreasing=TRUE)
```

The `catHigh2LowWithRefByEqualStats` contains the computed overlaps: for decreasing t-statistics (*i.e.* down-regulated genes):

```
> str(catHigh2LowWithRefByEqualStats)

List of 2
 $ dataSetA.t.vs.dataSetB.t:List of 2
  ..$ cat: num [1:506] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ deg: num [1:506] 2 2 2 2 2 2 2 2 2 2 ...
 $ dataSetA.t.vs.dataSetC.t:List of 2
  ..$ cat: num [1:506] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ deg: num [1:506] 2 2 2 2 2 2 2 2 2 2 ...
```

3.3 Computing Probability Intervals

The `calcHypPI` R function enables to compute the probability intervals for the CAT curves **obtained using equal ranks**. Such intervals are based on the hypergeometric distribution (see Irizarry et al [1] and Ross et al. [2]). Briefly, the `calcHypPI` uses `qhyper` quantile function to compute the **expected** proportions of common features between two ordered vectors of identifiers for a set of specified quantiles of the hypergeometric distribution.

To understand the way such proportions are obtained we can use the analogy of drawing a certain number of balls from an urn containing black and white balls, where black represents a failure (the vectors are in different order, and therefore the features do not overlap), and white represent a success (the vectors are in the same order, and hence the features overlap). According to this analogy the `calcHypPI` function uses the total number of features in common between the vectors as the total number of balls in the urn, and the size of the vector being compared as the number of balls drawn from the urn. Thus increasing vectors sizes (1, 2, 3, and so on until all features are used) correspond to an increasing number of balls drawn from the urn at each attempt.

By default the `calcHypPI` function assumes that the top 10% of the features of the two vectors are similarly ordered. In our analogy, therefore, the number of white balls in the urn corresponds to 10% of the total common features between the vectors. This expectation can be modified by the user with the `expectedProp` argument. When `expectedProp` is set equal to `NULL` the number of white balls in the urn (i.e. the top ranking features in the correct order) corresponds to the number of balls that are drawn at each attempt (i.e. the increasing size of top features from each vector that are being compared).

The example below shows how to compute probability intervals for CAT curves, using the default 0.1 expected proportion of top ranked features:

```
> PIbyRefEqualRanks <- calcHypPI(data=data)
```

The `PIbyRefEqualRanks` contains the computed probability intervals for the CAT curves:

```
> head(PIbyRefEqualRanks)

       0.999999 0.000001   0.999000 0.001000   0.990000 0.010000   0.950000 0.050000
[1,] 0.01960784        0 0.01960784        0 0.01960784        0 0.01960784        0
[2,] 0.03921569        0 0.03921569        0 0.01960784        0 0.01960784        0
[3,] 0.05882353        0 0.03921569        0 0.03921569        0 0.01960784        0
[4,] 0.07843137        0 0.05882353        0 0.03921569        0 0.03921569        0
[5,] 0.09803922        0 0.05882353        0 0.03921569        0 0.03921569        0
[6,] 0.09803922        0 0.07843137        0 0.05882353        0 0.03921569        0
     0.500000
[1,]        0
[2,]        0
[3,]        0
[4,]        0
[5,]        0
[6,]        0
```

The example below shows how to compute probability intervals for CAT curves, setting the expected proportion of top ranked features to 0.3:

```
> PIbyRefEqualRanks03 <- calcHypPI(data=data, expectedProp=0.3)
```

The `PIbyRefEqualRanks03` contains the computed probability intervals for the CAT curves:

```
> head(PIbyRefEqualRanks03)

        0.999999 0.000001    0.999000 0.001000    0.990000 0.010000    0.950000
[1,] 0.006578947        0 0.006578947        0 0.006578947        0 0.006578947
[2,] 0.013157895        0 0.013157895        0 0.013157895        0 0.013157895
[3,] 0.019736842        0 0.019736842        0 0.019736842        0 0.013157895
[4,] 0.026315789        0 0.026315789        0 0.019736842        0 0.019736842
[5,] 0.032894737        0 0.032894737        0 0.026315789        0 0.019736842
[6,] 0.039473684        0 0.032894737        0 0.032894737        0 0.026315789
     0.050000    0.500000
[1,]        0 0.000000000
[2,]        0 0.006578947
[3,]        0 0.006578947
[4,]        0 0.006578947
[5,]        0 0.006578947
[6,]        0 0.013157895
```

The example below shows how to compute probability intervals for CAT curves, using the default 0.1 expected proportion of top ranked features, for a set of specific hypergeometric distribution quantiles:

```
> PIbyRefEqualRanksQuant <- calcHypPI(data=data, prob=c(0.75, 0.9, 0.95, 0.99) )
```

The `PIbyRefEqualRanksQuant` contains the computed probability intervals for the CAT curves:

```
> head(PIbyRefEqualRanksQuant)
           0.75 0.25       0.90 0.10       0.95 0.05       0.99 0.01 0.50
[1,] 0.00000000    0 0.01960784    0 0.01960784    0 0.01960784    0    0
[2,] 0.00000000    0 0.01960784    0 0.01960784    0 0.01960784    0    0
[3,] 0.01960784    0 0.01960784    0 0.01960784    0 0.03921569    0    0
[4,] 0.01960784    0 0.01960784    0 0.03921569    0 0.03921569    0    0
[5,] 0.01960784    0 0.01960784    0 0.03921569    0 0.03921569    0    0
[6,] 0.01960784    0 0.03921569    0 0.03921569    0 0.05882353    0    0
```

The example below shows how to compute probability intervals for CAT curves, without specifying a proportion of expected top ranked features:

```
> PIbyRefEqualRanksNoExpectedProp <- calcHypPI(data=data, expectedProp=NULL)
```

The `PIbyRefEqualRanksNoExpectedProp` contains the computed probability intervals for the CAT curves:

```
> head(PIbyRefEqualRanksNoExpectedProp)
      0.999999 0.000001  0.999000 0.001000  0.990000 0.010000  0.950000 0.050000
[1,] 1.0000000        0 1.0000000        0 0.0000000        0 0.0000000        0
[2,] 1.0000000        0 0.5000000        0 0.0000000        0 0.0000000        0
[3,] 0.6666667        0 0.3333333        0 0.3333333        0 0.0000000        0
[4,] 0.5000000        0 0.2500000        0 0.2500000        0 0.0000000        0
[5,] 0.6000000        0 0.2000000        0 0.2000000        0 0.0000000        0
[6,] 0.5000000        0 0.3333333        0 0.1666667        0 0.1666667        0
     0.500000
[1,]        0
[2,]        0
[3,]        0
[4,]        0
[5,]        0
[6,]        0
```

4 Plotting the results

The `plotCat` function can be used to plot the CAT curves along with probability intervals, as derived using the `calcHypPI` function (see Figures 1 and 2, for CAT curves computed using a reference ranking, Figure 3 for examples in which the reference was not used, and Figure 4 for CAT curves for which the proportion of expected top ranked features was not specified).

4.1 CAT curves based on equal ranks using a reference.

The example below shows how to plot CAT curves based on equal ranks, along with probability intervals based on a fixed expected proportion of similarly ranked features of 30%, using the data set A as the reference.

```
> plotCat(catData = catHigh2LowWithRefByEqualRanks,
          preComputedPI=PIbyRefEqualRanks03,
          cex=1.2, lwd=1.2, cexPts=1.2, spacePts=30, col=c("red", "blue"),
          main="CAT curves for decreasing t-statistics",
          where="center", legend=TRUE, legCex=1, ncol=1,
          plotLayout = layout(matrix(1:2, ncol = 2), widths = c(2,1)))
[1] "Recycling colors"
```

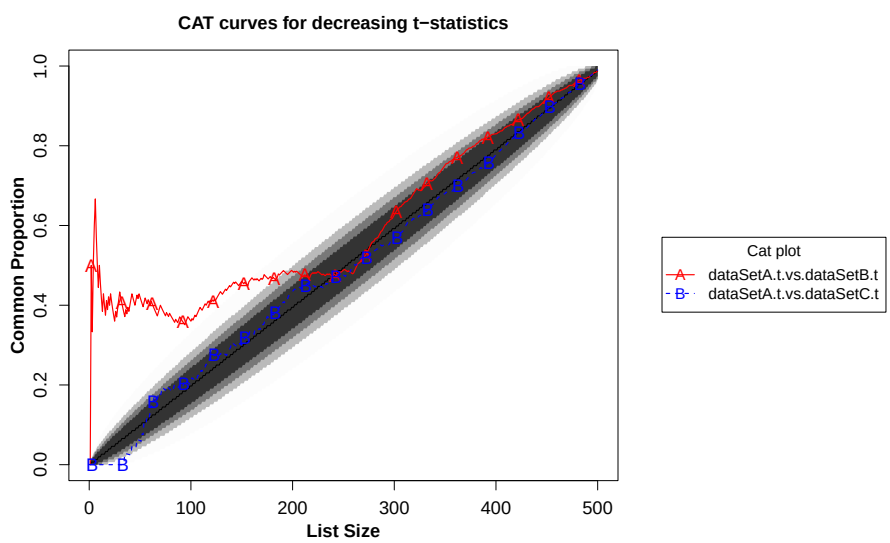

Figure 1: CAT-plot for curves based on equal ranks, using data set A as the reference. Lighter to darker grey shades represent probability intervals for distinct quantiles of the hypergeometric distribution (0.999999, 0.999, 0.99, 0.95), assuming 30% of the features to be similarly ranked.

4.2 CAT curves based on equal statistics using a reference.

The example below shows how to plot CAT curves based on equal statistics, using the data set A as the reference. When equal ranks are used each CAT curve has its own probability intervals, which therefore cannot be shown on the plot.

```
> plotCat(catData = catHigh2LowWithRefByEqualStats,
          cex=1.2, lwd=1.2, cexPts=1.2, spacePts=30, col=c("red", "blue"),
          main="CAT curves for decreasing t-statistics",
          where="center", legend=TRUE, legCex=1, ncol=1,
          plotLayout = layout(matrix(1:2, ncol = 2), widths = c(2,1)))
[1] "Recycling colors"
```

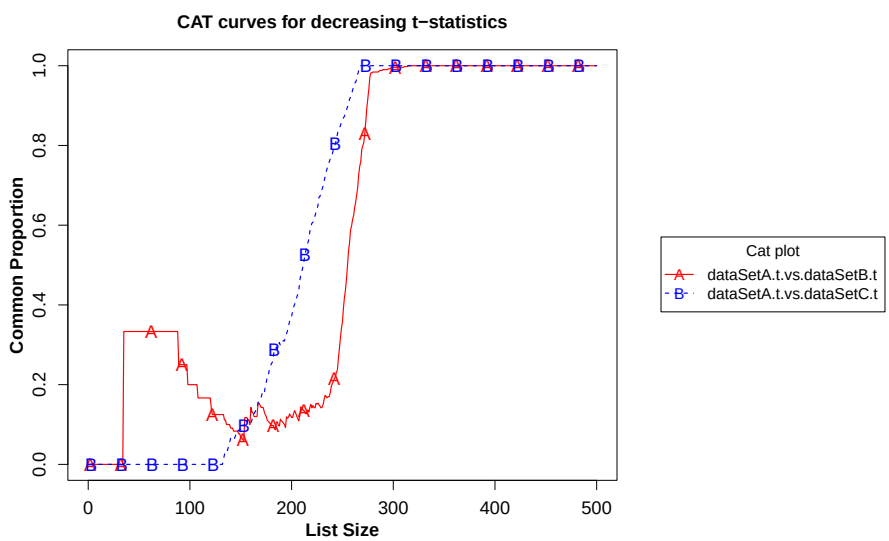

Figure 2: CAT-plot for curves based on equal t-statistics, using data set A as the reference.

4.3 CAT curves based on equal ranks without using a reference.

The example below shows how to plot CAT curves based on equal ranks, along with probability intervals based on a fixed expected proportion of similarly ranked features of 10%, without using any data set as the reference (all pair combinations are shown).

```
> plotCat(catData = catHigh2LowNoRefByEqualRanks,
          preComputedPI=PIbyRefEqualRanks,
          cex=1.2, lwd=1.2, cexPts=1.2, spacePts=30,
          main="CAT curves for decreasing t-statistics",
          where="center", legend=TRUE, legCex=1, ncol=1,
          plotLayout = layout(matrix(1:2, ncol = 2), widths = c(2,1)))
```

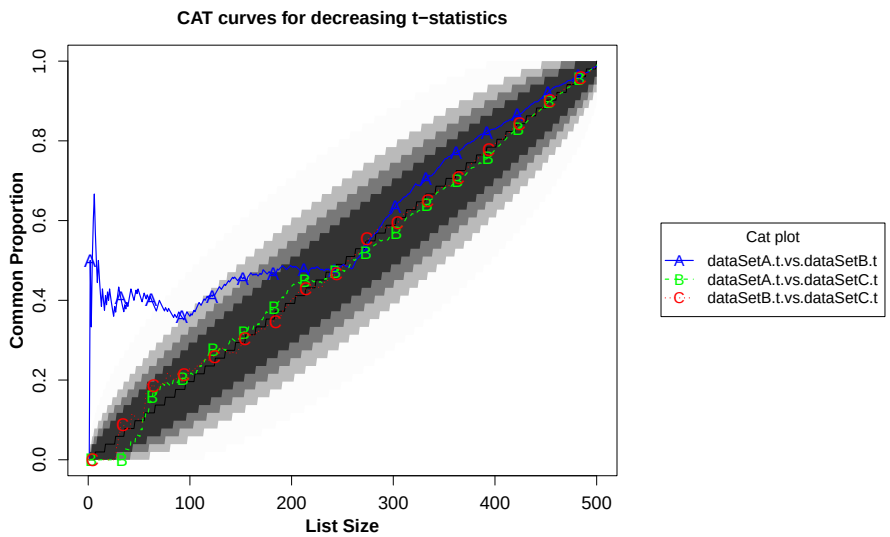

Figure 3: CAT-plot for curves based on equal ranks for all possible combinations of vectors. Lighter to darker grey shades represent probability intervals for the distinct quantiles of the hypergeometric distribution (0.999999, 0.999, 0.99, 0.95), assuming 10% of the features to be similarly ranked.

4.4 CAT curves based on equal ranks, unspecified expected proportion of corresponding features.

The example below shows how to plot CAT curves based on equal ranks, along with probability intervals based on an unspecified expected proportion of similarly ranked features, without using any data set as the reference (all pair combinations are shown).

```
> plotCat(catData = catHigh2LowNoRefByEqualRanks,
          preComputedPI=PIbyRefEqualRanksNoExpectedProp,
          cex=1.2, lwd=1.2, cexPts=1.2, spacePts=30,
          main="CAT curves for decreasing t-statistics",
          where="center", legend=TRUE, legCex=1, ncol=1,
          plotLayout = layout(matrix(1:2, ncol = 2), widths = c(2,1)))
```

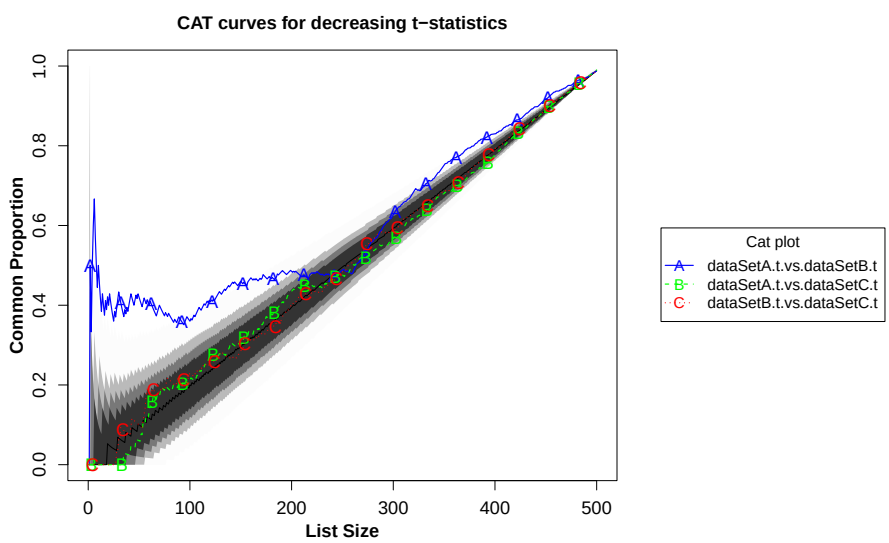

Figure 4: CAT-plot for curves based on equal ranks for all possible combinations of vectors. Lighter to darker grey shades represent probability intervals for distinct quantiles of the hypergeometric distribution (0.999999, 0.999, 0.99, 0.95). No expected proportion of similarly ranked features is specified.

5 System Information

Session information:

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

- R version 4.5.2 (2025-10-31), `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`
- Time zone: `Etc/UTC`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 24.04.3 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so` ; LAPACK version3.12.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: matchBox 1.53.0
- Loaded via a namespace (and not attached): buildtools 1.0.0, cli 3.6.5, compiler 4.5.2, evaluate 1.0.5, knitr 1.51, maketools 1.3.2, rlang 1.1.7, sys 3.4.3, tools 4.5.2, xfun 0.56

6 Literature Cited

References

[1] Rafael A Irizarry, Daniel Warren, Forrest Spencer, Irene F Kim, Shyam Biswal, Bryan C Frank, Edward Gabrielson, Joe G N Garcia, Joel Geoghegan, Gregory Germino, Constance Griffin, Sara C Hilmer, Eric Hoffman, Anne E Jedlicka, Ernest Kawasaki, Francisco Martínez-Murillo, Laura Morsberger, Hannah Lee, David Petersen, John Quackenbush, Alan Scott, Michael Wilson, Yanqin Yang, Shui Qing Ye, and Wayne Yu. Multiple-laboratory comparison of microarray platforms. *Nat Methods*, 2(5):345–350, May 2005.

[2] Ashley E Ross, Luigi Marchionni, Milena Vuica-Ross, Chris Cheadle, Jinshui Fan, David M Berman, and Edward M Schaeffer. Gene expression pathways of high grade localized prostate cancer. *The Prostate*, February 2011.

[3] B Benassi, R Flavin, L Marchionni, S Zanata, Y Pan, D Chowdhury, M Marani, S Strano, P Muti, and G Blandino. c-Myc is activated via USP2a-mediated modulation of microRNAs in prostate cancer. *Cancer Discovery*, 2012.