The core ncdDetect functionality

Malene Juul

24 August 2016

ncdDetect example runs

The package `ncdDetectTools` contains the core functions used in the driver detection method ncdDetect. The function `ncdDetect` is developed to perform convolution, i.e. to calculate the density function of the sum of independent discrete stochastic random variables.

The needed input to the `ncdDetect` function are the the matrices `predictions`, `scores` and `observations` (the latter is optional) which all have the same dimensionality. Each row represents a random variable, and each column represents a specific outcome of the variable.

The matrix `predictions` contains the probabilities for the individual stochastic variables. Each row must sum to one. The matrix `scores` contains the corresponding scores and the matrix `observations` contains 1 in the fields that are observed, and 0 in the rest. Each row must contain precisely one 1.

```
library(ncdDetectTools)
library(ggplot2)
library(poibin)

# import data to run examples
data("example_data")
```

Example 1: Throw two dice and sum up the eyes

Let X and Y be the outcomes of two throws with a die. Let S be the sum of these outcomes. We calculate the density function of S with `ncdDetect`:

```
# create matrices with :
# (a) probabilities for a die showing 1-6 - each row corresponds to a die
# (b) associated scores - in this case a die showing x will give a score of x
# (c) observations - assume that the first die show 3 and the second show 5

(throwTwoDice <- example_data$throwTwoDice)
```

```
## $predictions
##              1         2         3         4         5         6
## [1,] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
## [2,] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
##
## $scores
##      1 2 3 4 5 6
## [1,] 1 2 3 4 5 6
## [2,] 1 2 3 4 5 6
##
## $observations
##      1 2 3 4 5 6
## [1,] 0 0 1 0 0 0
## [2,] 0 0 0 0 1 0
```

```
# calculate the density function of S = X + Y and the p-value for the observed sum of 8
ncdDetect(predictions = throwTwoDice$predictions,
          scores = throwTwoDice$scores,
          observations = throwTwoDice$observations)
```

```
## $score_dist
##       y probability
##  1:  2  0.02777778
##  2:  3  0.05555556
##  3:  4  0.08333333
##  4:  5  0.11111111
##  5:  6  0.13888889
##  6:  7  0.16666667
##  7:  8  0.13888889
##  8:  9  0.11111111
##  9: 10  0.08333333
## 10: 11  0.05555556
## 11: 12  0.02777778
##
## $obs_score
## [1] 8
##
## $p_value
## [1] 0.4166667
```

Example 2: The binomial case

Assume that the four random variables X_1, X_2, X_3 and X_4 each follow a Bernoulli(0.2) distribution. Then $Y = X_1 + X_2 + X_3 + X_4 \sim$ Binomial(4,0.2). In the following, we calculate the density function directly using ncdDetect, and compare the result with what is obtained using the `pbinom()` function.

```
# create matrices with :
# (a) probabilities - each row corresponds to one of the four random variables
# (b) associated scores - these are 1 and 0 (outcomes of Bernoulli r.v.)

(binomialExampleSmall <- example_data$binomialExampleSmall)
```

```
## $predictions
##        0   1
## [1,] 0.8 0.2
## [2,] 0.8 0.2
## [3,] 0.8 0.2
## [4,] 0.8 0.2
##
## $scores
##      0 1
## [1,] 0 1
## [2,] 0 1
## [3,] 0 1
## [4,] 0 1
```

```
# calculate the density function of Y = X1 + X2 + X3 + X4
(ncdDetect_output <- ncdDetect(predictions = binomialExampleSmall$predictions,
                               scores = binomialExampleSmall$scores))
```

```
## $score_dist
##    y probability
## 1: 0      0.4096
## 2: 1      0.4096
## 3: 2      0.1536
## 4: 3      0.0256
## 5: 4      0.0016
```

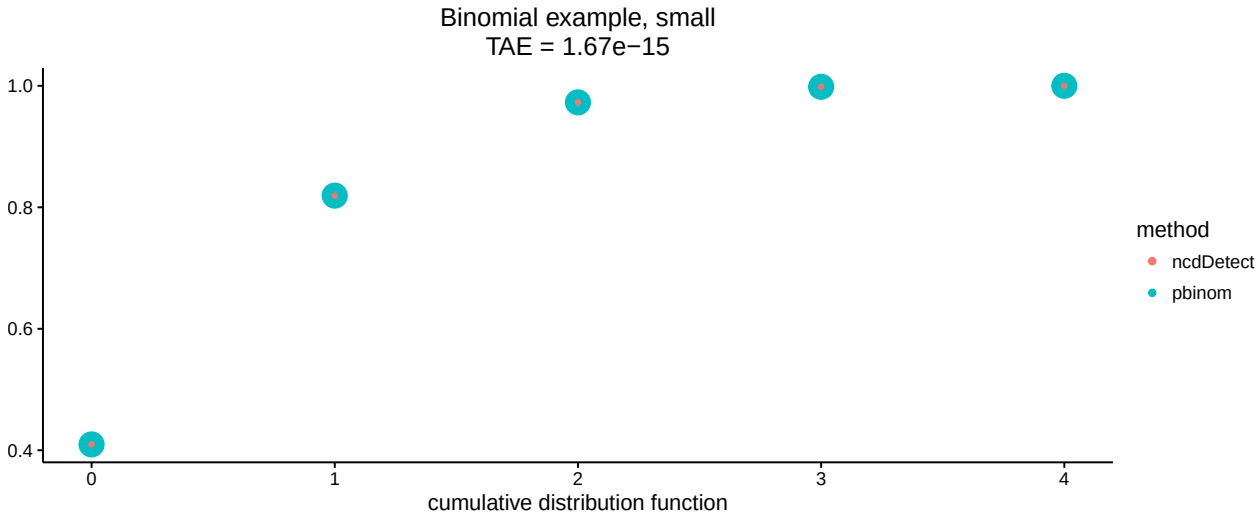
```
# calulate the cdf
ncdDetect_output$score_dist[, ncdDetect_cdf := cumsum(probability)]
ncdDetect_output$score_dist[, probability := NULL]

# compare to the results from pbinom() function
binomial_result <- data.table(y = 0:4,
                              binom_cdf = pbinom(q = 0:4, size = 4, prob = 0.2))

setkey(binomial_result, y)
setkey(ncdDetect_output$score_dist, y)
(comparison <- binomial_result[ncdDetect_output$score_dist])
```

```
##    y binom_cdf ncdDetect_cdf
## 1: 0    0.4096        0.4096
## 2: 1    0.8192        0.8192
## 3: 2    0.9728        0.9728
## 4: 3    0.9984        0.9984
## 5: 4    1.0000        1.0000
```

```
# calculate total absolute error (TAE) between the two cdfs
(TAE <- sum(abs(comparison[,binom_cdf - ncdDetect_cdf])))
```

```
## [1] 1.665335e-15
```

A comparison of the cumulative distribution functions (cdf) is shown in the below figure.



Example 3: The binomial case, continued

Assume that the 500 random variables $X_1, \ldots, X_{500}$ each follow a Bernoulli(0.2) distribution. Then $Y = X_1 + \ldots + X_{500} \sim$ Binomial(500,0.2). In the following, we calculate the density function directly using ncdDetect, and compare the result with what is obtained using the `pbinom()` function.

Note that if we're only interested in the output distribution up to a certain value, we can set a threshold to save computations. In this case, we only calculate the density function for $P(X = x), x \in \{0, \ldots, 250\}$. The output is still a density function, but the probability mass of $P(X > 250)$ is aggregated together in $P(X = 251)$ This feature can be convenient to avoid calculating probabilties of a potential uninteresting large tail.

```
# create matrices with :
# (a) probabilities - each row corresponds to one of the 500 random variables
# (b) associated scores - these are 1 and 0 (outcomes of Bernoulli r.v.)

binomialExampleLarge <- example_data$binomialExampleLarge
as.data.table(binomialExampleLarge$predictions)
```

```
##        0   1
##   1: 0.8 0.2
##   2: 0.8 0.2
##   3: 0.8 0.2
##   4: 0.8 0.2
##   5: 0.8 0.2
##  ---
## 496: 0.8 0.2
## 497: 0.8 0.2
## 498: 0.8 0.2
## 499: 0.8 0.2
## 500: 0.8 0.2
```

```
as.data.table(binomialExampleLarge$scores)
```

```
##      0 1
##   1: 0 1
##   2: 0 1
##   3: 0 1
##   4: 0 1
##   5: 0 1
##  ---
## 496: 0 1
## 497: 0 1
## 498: 0 1
## 499: 0 1
## 500: 0 1
```

```
# calculate the density function of Y = X1 + ... + X500 with threshold = 250
(ncdDetect_output <- ncdDetect(predictions = binomialExampleLarge$predictions,
                               scores = binomialExampleLarge$scores,
                               thres = 250 + 1))
```

```
## $score_dist
```

```
##         y  probability
##   1:    0 3.507466e-49
##   2:    1 4.384333e-47
##   3:    2 2.734728e-45
##   4:    3 1.134912e-43
##   5:    4 3.525320e-42
##  ---
## 248:  247 7.723384e-49
## 249:  248 1.969774e-49
## 250:  249 4.983767e-50
## 251:  250 1.250925e-50
## 252:  251 3.114854e-51
```

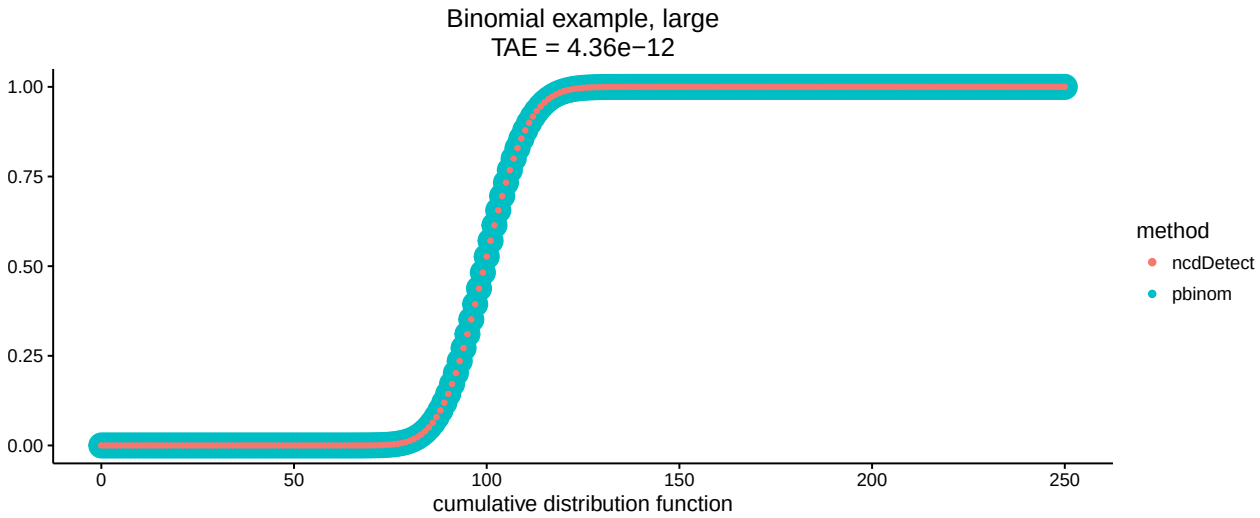
```
# calulate the cdf
ncdDetect_output$score_dist[, ncdDetect_cdf := cumsum(probability)]
ncdDetect_output$score_dist[, probability := NULL]
ncdDetect_output$score_dist <- ncdDetect_output$score_dist[y <= 250,]

# compare to the results from pbinom() function
binomial_result <- data.table(y = 0:250,
                              binom_cdf = pbinom(q = 0:250, size = 500, prob = 0.2))
setkey(binomial_result, y)
setkey(ncdDetect_output$score_dist, y)
(comparison <- binomial_result[ncdDetect_output$score_dist])
```

```
##         y    binom_cdf ncdDetect_cdf
##   1:    0 3.507466e-49  3.507466e-49
##   2:    1 4.419407e-47  4.419407e-47
##   3:    2 2.778922e-45  2.778922e-45
##   4:    3 1.162701e-43  1.162701e-43
##   5:    4 3.641590e-42  3.641590e-42
##  ---
## 247:  246 1.000000e+00  1.000000e+00
## 248:  247 1.000000e+00  1.000000e+00
## 249:  248 1.000000e+00  1.000000e+00
## 250:  249 1.000000e+00  1.000000e+00
## 251:  250 1.000000e+00  1.000000e+00
```

```
# calculate total absolute error (TAE) between the two cdfs
(TAE <- sum(abs(comparison[,binom_cdf - ncdDetect_cdf])))
```

```
## [1] 4.35781e-12
```

A comparison of the cdfs is shown in the below figure.



Example 4: The Poisson-binomial case

Assume that the random variables X_i, $i \in \{1, \ldots, 1000\}$ follow a Bernoulli(p_i) distribution. Then $Y = \sum_{i=1}^{1000} X_i$ follow a Poisson-binomial distribution. In the following, we calculate the density function directly using ncdDetect, and compare the result with what is obtained using the `ppoibin()` function from the `poibin` R-package (https://cran.r-project.org/web/packages/poibin/index.html).

```
# create matrices with :
# (a) probabilities - each row corresponds to one of the 1,000 random variables
# (b) associated scores - these are 1 and 0 (outcomes of Bernoulli r.v.)

poissonBinomialExample <- example_data$poissonBinomialExample
as.data.table(poissonBinomialExample$predictions)
```

```
##                 1          0
##    1: 0.113703411 0.88629659
##    2: 0.622299405 0.37770060
##    3: 0.609274733 0.39072527
##    4: 0.623379442 0.37662056
##    5: 0.860915384 0.13908462
##   ---
##  996: 0.001308702 0.99869130
##  997: 0.767425936 0.23257406
##  998: 0.319979546 0.68002045
##  999: 0.958012845 0.04198716
## 1000: 0.195397579 0.80460242
```

```
as.data.table(poissonBinomialExample$scores)
```

```
##       1 0
##    1: 1 0
##    2: 1 0
##    3: 1 0
##    4: 1 0
##    5: 1 0
```

```
##   ---
##  996: 1 0
##  997: 1 0
##  998: 1 0
##  999: 1 0
## 1000: 1 0
```

```
# calculate the density function of Y = X1 + ... + X1000
(ncdDetect_output <- ncdDetect(predictions = poissonBinomialExample$predictions,
                               scores = poissonBinomialExample$scores))
```

```
## $score_dist
##        y   probability
##   1:  55 9.881313e-324
##   2:  56 6.521667e-322
##   3:  57 4.214874e-320
##   4:  58 2.666107e-318
##   5:  59 1.645285e-316
##  ---
## 899: 953 1.470971e-316
## 900: 954 1.988002e-318
## 901: 955 2.604220e-320
## 902: 956 3.310240e-322
## 903: 957 4.940656e-324
```

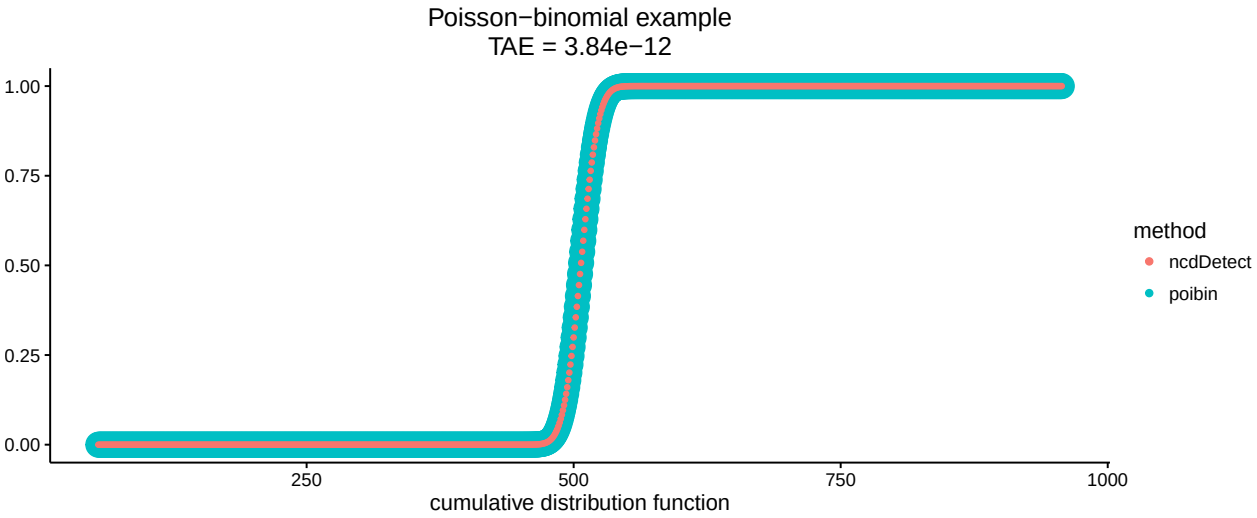
```
# calulate the cdf
ncdDetect_output$score_dist[, ncdDetect_cdf := cumsum(probability)]
ncdDetect_output$score_dist[, probability := NULL]

# compare to the results from ppoibin() function
poibin_result <- data.table(y = 0:1000,
                            poibin_cdf = ppoibin(kk = 0:1000,
                                                 pp = poissonBinomialExample$predictions[,1],
                                                 method = "DFT-CF"))
setkey(poibin_result, y)
setkey(ncdDetect_output$score_dist, y)
(comparison <- poibin_result[ncdDetect_output$score_dist])
```

```
##        y   poibin_cdf ncdDetect_cdf
##   1:  55 7.052741e-15 9.881313e-324
##   2:  56 7.090562e-15 6.620480e-322
##   3:  57 7.129492e-15 4.281079e-320
##   4:  58 7.172747e-15 2.708917e-318
##   5:  59 7.220550e-15 1.672374e-316
##  ---
## 899: 953 1.000000e+00  1.000000e+00
## 900: 954 1.000000e+00  1.000000e+00
## 901: 955 1.000000e+00  1.000000e+00
## 902: 956 1.000000e+00  1.000000e+00
## 903: 957 1.000000e+00  1.000000e+00
```

```
# calculate total absolute error (TAE) between the two cdfs
(TAE <- sum(abs(comparison[,poibin_cdf - ncdDetect_cdf])))
```

```
## [1] 3.839603e-12
```

A comparison of the cdfs is shown in the below figure.



Example 5: Adding discrete random variables with different sizes of outcome space

In the above examples, `ncdDetect` has been applied to random variables with identical sizes of outcome space. By using the underlying function `convolution()`, it is possible to convolute discrete random variables of different dimensions. The input data has a slightly different format, as described in the example below in which we add three discrete random variables.

```
# create a data.table with columns
#   x - a numeric indicator unique for each random variable
#   y - outcome value (score)
#   probability - probability of the associated y
# the subset of the data.table with a unique value of x corresponds to a random variable,
# and the probabilities within each x must thus sum to 1

# we consider three random variables (x = 1, 2, 3),
# with densities given by columns "y" and "probability":
(generalConvolution <- example_data$generalConvolution)
```

```
##    x y probability
## 1: 1 1   0.7500000
## 2: 1 2   0.2500000
## 3: 2 1   0.3333333
## 4: 2 2   0.3333333
## 5: 2 3   0.3333333
## 6: 3 1   0.1000000
## 7: 3 2   0.2000000
## 8: 3 3   0.3000000
## 9: 3 4   0.4000000
```

```
convolution(generalConvolution)
```

```
##    y probability
## 1: 3  0.02500000
## 2: 4  0.08333333
## 3: 5  0.17500000
## 4: 6  0.27500000
## 5: 7  0.25000000
## 6: 8  0.15833333
## 7: 9  0.03333333
```

Session information

```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.6 (El Capitan)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] data.table_1.9.6   poibin_1.2         ggplot2_2.1.0
## [4] ncdDetectTools_1.0
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.6              digest_0.6.10
##  [3] plyr_1.8.4               chron_2.3-47
##  [5] grid_3.3.1               gtable_0.2.0
##  [7] formatR_1.4              magrittr_1.5
##  [9] scales_0.4.0             evaluate_0.9
## [11] stringi_1.1.1            RcppArmadillo_0.7.200.2.0
## [13] rmarkdown_1.0            labeling_0.3
## [15] tools_3.3.1              stringr_1.0.0
## [17] munsell_0.4.3            yaml_2.1.13
## [19] colorspace_1.2-6         htmltools_0.3.5
## [21] knitr_1.13
```