TopKLists: Analyzing multiple ranked lists

Michael G. Schimek, Eva Budinska, Jie Ding, Karl G. Kugler,
Vendula Svendova and Shili Lin

May 28, 2020

Contents

1 Introduction

The ranking of distinct items has become mainstream in recent years. Examples include Web search engine results for query terms, institution league tables in higher education, preference rankings of brands, betting results in sports, results from microarray platforms in biotechnology, and meta-analysis of multiple study findings in medicine, among many others. The rank position of an object or institution might be the result of measuring the strength of evidence or of assessment based on expert knowledge or preference. The assessors can be a person or a technical device. Typically, such lists are between several thousand and several tens of thousands of items in length. However, only a comparably small subset of k top-ranked items is usually informative. These are characterized by a strong overlap of their rank positions when they are ranked by several independent assessors.
There are two basic statistical tasks: (i) Identification of the top-k most conforming items. For this task, two lists are analyzed together (multiple lists in a pairwise manner). (ii) Calculation of a consolidated top-k sublist with a new optimized ordering of the conforming items from two or more lists. For long ranked lists, (i) is a prerequisite of (ii) because for any kind of rank aggregation the top-k list lengths of the individual top-k lists need to be specified.
For these two tasks this package offers several options which can be selected from three modules that are provided. Various options of high practical value are supported by a graphical user interface (GUI):

1. `TopKInference` provides exploratory nonparametric inference for the estimation of the top k list length of paired rankings;

2. `TopKSpace` provides several rank aggregation techniques (Borda, Markov Chain, and Cross Entropy Monte Carlo) which allow for the combination of input lists of different lengths, that may come from different underlying sets (spaces);

3. `TopKGraphics` provides a collection of graphical tools for the visualization of the inputs to, and the outputs from, the other modules.

Whenever you refer to the package `TopKLists`, please cite [2], [3] and [6]. In the following sections, we illustrate the usage of these modules.

1.1 Input data

The input data should be either lists of ranked object names (see the breast cancer example below), or lists of their actual rank positions. Data should be prepared in a `data.frame` format (for the truncation functions `compute.stream`, `j0.multi`, `TopKListsGUI`, `deltaplot`,...) or a `list` format (for the aggregation functions `Borda`, `MC` and `CEMC`). The truncation functions allow for incomplete data sets (NA values). After loading the data, user can continue working in the console using all the functions directly, or start the graphical user interface (GUI) by running the `TopKListsGUI` function.

1.2 The breast cancer data

For illustrating the usage of this package, we selected three breast cancer microarray datasets (MDCC, TransBig and Pusztai - each list of the length 917) that include information on estrogen receptor positivity (ER+ vs ER-), which is one of the prognostic markers for breast cancer (for the accession numbers see Table 1):

NAME	PubMed ID	GEO	Reference
MDCC	20676074	GSE20194	[8]
TransBig	17545524	GSE7390	[1]
Pusztai	20829329	GSE20271	[9]

Table 1: Example dataset: names, PubMed IDs, GEO accession numbers and references

The presence of an estrogene receptor on cancer cells discriminates between two groups of breast cancer and therefore we expected ESR1 (estrogen receptor 1 gene) to be in the first place among the differentially expressed genes. Given the study designs, we also expected a substantial overlap in the top ranked genes. We begin by loading the data set and displaying the first part of the data frame, as described below.

```
library(TopKLists)
data(breast)
head(breast)
```

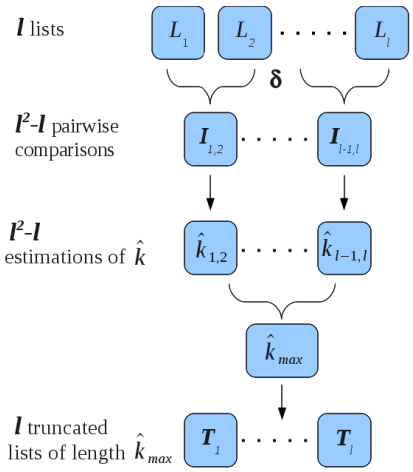


Figure 1: The inference concept to obtain ℓ truncated consensual lists T_i from ℓ full ranked lists L_i

```
##   TransBig   MDCC Pusztai
## 1     ESR1   ESR1    ESR1
## 2   TBC1D9 TBC1D9  TBC1D9
## 3   SCUBE2    EVL  SCUBE2
## 4      EVL SCUBE2    FBP1
## 5     NAT2  CIRBP     EVL
## 6     BTG2   FBP1    RHOB
```

2 Methods

2.1 TopKInference

The nonparametric inference method of [2] for the truncation of paired ranked lists forms the core part of the `TopKInference` module. The associated iterative algorithm, as implemented here, allows the estimation of the length, k, of a top-k list in the presence of irregular and missing assignments. Overlap of rank positions in two input lists is represented by a sequence of indicators, I, where $I_j = 1$ is the ranking, given by the second assessor to the object ranked j by the first assessor, is not more than δ index positions distant from j, and otherwise $I_j = 0$. The vector of indicators is represented by the `Idata` variable. The variables I_j are assumed to follow a Bernoulli random distribution. This implies independence, which is motivated by $k \ll N$ and a strong random contribution due to irregular assignments in real data. However, [2] could prove that their theoretical results obtained under the assumption of complete independence also applies in the situation of moderate m-dependence. Simulation study evidence (unpublished material) is in full support of these theoretical findings: in fact, even under substantial list dependence, stable truncation results can be obtained from `TopKInference`.
The algorithmic solution employed by `TopKInference` in the event of there being more than two ranked lists is outlined in Figure 1. There, the principle for the calculation of an overall index k^* (a function of the individual k's from the ℓ lists L_i; the maximum `maxK` is the default) based on all pairwise comparisons is outlined. Having obtained such an overall index, we arrive at truncated lists T_i. They can either be aggregated by graphical means (the `aggmap` of the `TopKGraphics` module) or by stochastic rank aggregation (`TopKSpace` module). Details of the approach taken for multiple lists can be found in [7].

2.1.1 Construction of a dataset and execution of the inference procedure

In order to run the examples in this vignette, the TopKLists package must first be loaded:

```
library(TopKLists)
```

The truncation point j_0, where noise takes over for a pair of ranked lists (i.e., the first index position after the end of the top-k list), can be estimated for any prespecified distance δ and the chosen tuning parameter values C and ν. It should be noted that in the R source code δ is denoted by d, C by const and ν by v. In this example we simulate a dataset with an assumed top length of $k = 30$, hence the truncation point of $j_0 = 31$.

```
k = 30
set.seed(123)
x = c(rep(1,k), rbinom(100, 1, 0.2))
x
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [28] 1 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 1
##  [55] 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
##  [82] 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0
## [109] 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
```

Now let us estimate j_0 for different values of ν using the `compute.stream` function. This outputs a list of estimated j_0 values, as well as other details of the algorithm's convergency.

```
v.vect=seq(2,length(x), by=2) #setting up a vector of the nu values

resF=c()

for (v in v.vect)
 {
        res=compute.stream(x, const=0.5, v)
        resF=rbind(resF,c(v,paste(res)))
 }

colnames(resF)=c("v", "j0_est", "k","reason.break", "Js", "v.vector")
head(resF)
```

```
##      v    j0_est k    reason.break
## [1,] "2"  "32"   "31" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
## [2,] "4"  "31"   "30" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
## [3,] "6"  "31"   "30" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
## [4,] "8"  "31"   "30" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
## [5,] "10" "30"   "29" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
## [6,] "12" "32"   "31" "Js[i-2]==Js[i] & Js[i-1]==Js[i-3]"
##      Js                        v.vector
## [1,] "c(31, 31, 31, 31, 31)" "c(2, 2, 2, 2, 2)"
## [2,] "c(29, 31, 29, 31, 29)" "c(4, 4, 4, 4, 4)"
## [3,] "c(28, 32, 28, 32, 28)" "c(6, 6, 6, 6, 6)"
## [4,] "c(27, 33, 27, 33, 27)" "c(8, 8, 8, 8, 8)"
## [5,] "c(25, 33, 25, 33, 25)" "c(10, 10, 10, 10, 10)"
## [6,] "c(26, 36, 26, 36, 26)" "c(12, 12, 12, 12, 12)"
```

```
table(resF[,2])
```

```
##
## 30 31 32 33 34 35 36 37
##  1  4 12  2  3 10 24  9
```

The `compute.stream` function outputs the index `j0est`, where the information between the lists degenerates into noise; index `k=j0est-1`; the sequence of estimated `j0s` in each run (J_s); the reason why computation has ended (`reason.break`); and the preselected value of the parameter ν (`v`).

2.1.2 Visualization of the truncation results

The following plot (Figure 2) summarizes the obtained estimation results $\hat{j}_0$ for the specified range of pilot sample sizes ν and the assumed point of degeneration $j_0 = 31$. Small values of ν are the most appropriate here.

```
plot(resF[,1], resF[,2], pch=19, ylim=c(25, 40),
xlab=substitute(nu), ylab=substitute(paste(hat(j)[0])))
abline(a=31, b=0, col="red")
lines(resF[,1], resF[,2])
```

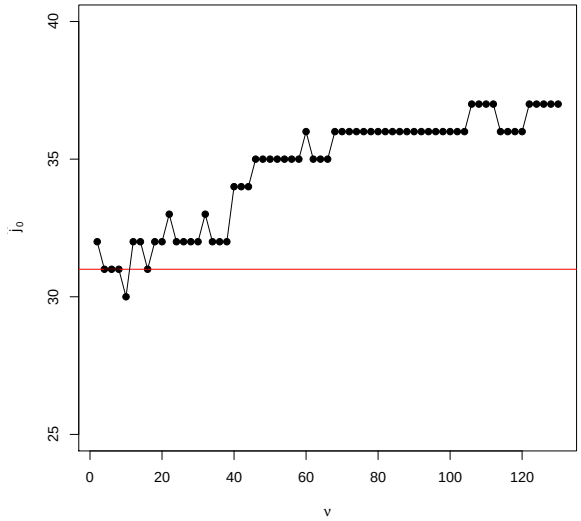


Figure 2: Estimation of j_0 for different values of ν

2.2 TopKSpace

The principle of the `TopKSpace` module is to consolidate information from the l top-k lists to arrive at an aggregate list, AL. As shown in Figure 3, the top-k lists $(L_1, L_2, \ldots, L_l)$ may not only be of different lengths, they may also come from studies or assessments that consider different sets of objects, hence the underlying spaces $(S_1, S_2, \ldots, S_l)$ from which the top-k lists are derived may actually be different. The goal of the inference in `TopKSpace` is to therefore find the top-k list, AL, from the aggregate new space $(\cup_{i=1}^{l} L_i)$, such that the weighted sum of distances between each of the input lists and AL will be the minimum among lists of the same length. Two distance measures, Kendall's τ and Spearman's footrule, are available in the package. Both take the differences in the underlying spaces into account [4]. There are three common assumptions about the underlying spaces: *common-space* (all top-k lists come from a single common space), *underlying space-dependent*, i.e. *assessment-* or *platform-dependent* (using the known spaces from which the top-k lists were generated), and *top-k-space* (treating each top-k list as its own space). Since underlying space-dependent represents the true underlying scenario, this method is recommended if such information is available.
There are three classes of algorithms implemented in `TopKSpace`, namely Borda's method, Markov Chain (MC) algorithms [5], and a Cross Entropy Monte Carlo (CEMC) method taking advantage of the new Order Explicit Algorithm (OEA) as described by [3]. The Borda and Markov Chain methods consist of heuristic algorithms which do not directly optimize the objective function (i.e., minimizing the weighted distances), whereas the CEMC method employs a Monte Carlo search procedure for achieving this optimization. Borda and Markov Chain algorithms run much faster than the Cross Entropy Monte Carlo algorithm, however the latter usually achieves better results. Nevertheless, simulation studies indicate that taking the underlying space into consideration has a much greater impact than using different algorithms. These three algorithms are implemented in three top-level functions within `TopKSpace`: `Borda`, `MC`, and `CEMC`. There are also two plotting functions: `Borda.plot` and `MC.plot` that help the user to visualize and compare the results.

2.2.1 Construction of three input lists and underlying spaces

Let us first produce the following three ranked lists of different lengths and their underlying spaces. Although this example is contrived, it is realistic in terms of the lengths and the number of top-k lists.

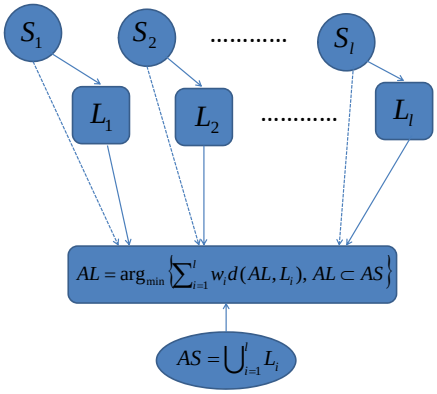


Figure 3: The optimization concept for the aggregation of l full ranked lists L_i into one list AL under space consideration

```
set.seed(1234)
L1=paste("Obj",1:30,sep="")
L2=paste("Obj",c(1:10,31:40,11:15),sep="")
L3=paste("Obj",c(1:10,16:20,11:15),sep="")
input=list(L1,L2,L3)
space1=space2=space3=paste("Obj",1:40,sep="")
space=list(space1,space2,space3)
```

As can be seen from the compositions of the three lists, they are of different lengths (30, 25, and 20 for L1, L2, and L3, respectively). Further, the aggregate candidate set is objects Obj1-Obj40, although two of the underlying spaces are larger, indicating that objects Obj41-Obj50 were not selected in the top-k of the two corresponding lists. Fifteen of these 40 objects appear in all 3 lists, with objects Obj1-Obj10 ranked 1-10, in that order, in all three lists; objects Obj11-Obj15, however, have different rankings (they are ranked 11-15 for L1, 20-25 for L2, and 16-20 for L3). Five objects, Obj16-Obj20, appear in two of the lists, with rankings of 16-20 for L1 and 11-15 for L3. The remaining 20 objects, Obj21-Obj40, appear on only one list (objects Obj21-Obj30 are ranked 21-30 in L1, while objects Obj31-Obj40 are ranked 11-20 in L2). Since the number of potential aggregate top-40 lists is more than 8×10^{47}, enumerating all of them to find the "true" answer is not possible, but it would be reasonable to make a fairly accurate guess. There should be overwhelming information for ranking objects Obj1-Obj10 in the top-10. Objects Obj11-Obj15 should come next, followed by Obj16-Obj20. For the remaining of the objects, Obj31-Obj40 should be ranked ahead of Obj21-Obj30. In the following subsection, we will demonstrate the use of the three classes of algorithms (`Borda`, `MC`, `CEMC`) for aggregation of these three ranked lists. We will also demonstrate the use of an objective evaluation criterion to compare the relative performances of different algorithms.

2.2.2 Function calling

The Borda algorithm Four Borda scores are implemented in the `Borda` function of `TopKSpace`, namely the arithmetic mean (ARM), median (MED), geometric mean (GEO), and L2-norm (L2N). The output is a list with two elements: `TopK` provides the aggregate rankings and `Scores` gives the corresponding Borda scores for each of the four functions. Of the three arguments in `Borda`, only the first one (*input*) is required. If argument *space* is not provided, then the underlying space is assumed to be common among all top-k lists implicitly, and the union of the top-k lists is treated as the common space. If argument k is not specified, then the full ranked list (all elements in the union) is outputted.

```
outBorda=Borda(input,space)
# "space" is explicitly specified; underlying space-dependent
```

```
outBorda1=Borda(input)
#"space" is not specified; all lists are assumed to come from the common space (objects Obj1-Obj40)
```

```
outBorda2=Borda(input,space=input)
# "space = input" indicates that this is the top-k space
```

```
sum(outBorda$Scores-outBorda1$Scores)

## [1] 0

sum(outBorda$Scores-outBorda2$Scores)

## [1] 279.6785
```

Comment 1: Objects Obj1-Obj40, union of the three top-k lists, are contained in all three spaces. As such, the results for which the spaces were explicitly specified should be the same as those with the common space assumption, which was exactly what we saw above. On the other hand, the underlying space-dependent results were different from the results assuming top-k spaces (i.e., all lists were full ranked lists).

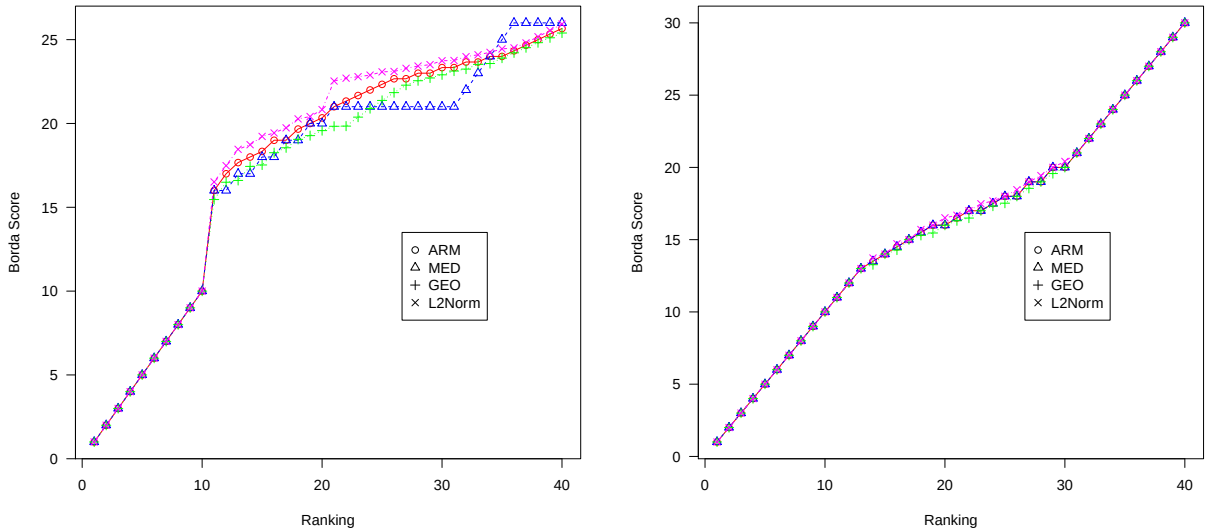
```
as.list(outBorda$TopK)

## $mean
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj16" "Obj13"
## [15] "Obj17" "Obj14" "Obj18" "Obj19" "Obj15" "Obj20" "Obj31"
## [22] "Obj32" "Obj33" "Obj34" "Obj35" "Obj21" "Obj36" "Obj22"
## [29] "Obj37" "Obj23" "Obj38" "Obj24" "Obj39" "Obj25" "Obj40"
## [36] "Obj26" "Obj27" "Obj28" "Obj29" "Obj30"
##
## $median
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj16" "Obj12" "Obj17"
## [15] "Obj13" "Obj18" "Obj14" "Obj19" "Obj15" "Obj20" "Obj21"
## [22] "Obj31" "Obj32" "Obj33" "Obj34" "Obj35" "Obj36" "Obj37"
## [29] "Obj38" "Obj39" "Obj40" "Obj22" "Obj23" "Obj24" "Obj25"
## [36] "Obj26" "Obj27" "Obj28" "Obj29" "Obj30"
##
## $geo.mean
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj16" "Obj17"
## [15] "Obj13" "Obj18" "Obj14" "Obj19" "Obj31" "Obj15" "Obj20"
## [22] "Obj32" "Obj33" "Obj34" "Obj35" "Obj36" "Obj37" "Obj21"
## [29] "Obj38" "Obj22" "Obj39" "Obj23" "Obj40" "Obj24" "Obj25"
## [36] "Obj26" "Obj27" "Obj28" "Obj29" "Obj30"
##
## $l2norm
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj13" "Obj16"
## [15] "Obj17" "Obj14" "Obj18" "Obj19" "Obj15" "Obj20" "Obj31"
## [22] "Obj32" "Obj21" "Obj33" "Obj34" "Obj22" "Obj35" "Obj23"
## [29] "Obj36" "Obj37" "Obj24" "Obj38" "Obj25" "Obj39" "Obj26"
## [36] "Obj40" "Obj27" "Obj28" "Obj29" "Obj30"
```

Comment 2: All four scoring functions ranked objects Obj1-Obj10, in that order, in the top-10. However, despite our expectation that objects Obj11-Obj15, in that order, should be ranked 11-15, followed by objects Obj16-Obj20, the results were more of a mix of these two groups, but with the correct relative orders preserved within each group. The results are quite similar among the four scoring functions. This is not surprising due to degradation of ranking information beyond the top-10. Such information degradation can be seen by plotting the Borda scores as described in the following.

Plotting the Borda scores against the ranking can frequently reveal when information for ranking starts to diminish. For example, Figure 4(a) shows that for the underlying space-dependent approach, a large gap appears between

the Borda scores at rankings 10–11, and that the changes of the scores also slow down after this point. This can be similarly observed for the scores under the top-k space assumption (Figure 4(b)).

```
Borda.plot(outBorda, k=40) # plot scores from underlying space-dependent analysis
Borda.plot(outBorda2, k=40) # plot scores from top-k space analysis
```



(a) Underlying space-dependent analysis

(b) Top-k space analysis

Figure 4: Plotting Borda's scores

Note that the first argument of the `Borda.plot` function is the output from the Borda function, which is mandatory. If the number of ranked objects (second argument) is not specified, the plot will show scores for all objects in the union set.

Three MC algorithms Three Markov Chain (MC) algorithms are implemented in the `MC` function in `TopKSpace`: MC1 (spam sensitive), MC2 (majority rule), and MC3 (proportional). Among them, `MC3` may be more appropriate for multi-platform omics problems given the potential of unique features of each data type. The output is a list with two elements for each of the `MC` algorithms (aggregate top-k list: `MC1.TopK`, `MC2.TopK`, or `MC3.TopK`; stationary probability distribution: `MC1.Prob`, `MC2.Prob`, or `MC3.Prob`). Of the arguments in `MC`, only the first one, *input*, is required. If argument *space* is not provided, it is assumed that the underlying space is common among all top-k lists, and the union of the top-k list is supplied as the common space. If argument k is not specified, then the full ranked list (all elements in the union) is outputted. The other two arguments (a and $delta$; the latter different from δ in `TopKInference`) are tuning parameters, whose default values were shown to work well in a number of examples.

```
outMC=MC(input,space)
# "space" is explicitly specified; underlying space-dependent
```

```
outMCa=MC(input,k=30)
# "space" is not specified, so it is the same as common space (01-040)
```

```
outMCb=MC(input,space=input)
# "space = input" indicates that this is the top-k space
```

```
sum(outMC$MC2.Prob-outMCa$MC2.Prob)

## [1] 0
```

Comment 1: Similar to the discussion for running the `Borda` function, the results for which the space was explicitly specified were expected to be the same as those obtained under the common space assumption, which was exactly what we saw above (the two stationary distributions were identical). On the other hand, the common-space results were different from those assuming top-k space (there were large discrepancies in the aggregated top-k lists).

```
list(outMC$MC1.TopK, outMC$MC2.TopK, outMC$MC3.TopK)

## [[1]]
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj16" "Obj11" "Obj17" "Obj12"
## [15] "Obj18" "Obj13" "Obj31" "Obj19" "Obj14" "Obj32" "Obj20"
## [22] "Obj15" "Obj33" "Obj34" "Obj35" "Obj36" "Obj37" "Obj21"
## [29] "Obj38" "Obj22" "Obj39" "Obj23" "Obj40" "Obj24" "Obj25"
## [36] "Obj26" "Obj27" "Obj28" "Obj29" "Obj30"
##
## [[2]]
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj13" "Obj14"
## [15] "Obj15" "Obj16" "Obj17" "Obj18" "Obj19" "Obj20" "Obj40"
## [22] "Obj39" "Obj38" "Obj37" "Obj36" "Obj35" "Obj34" "Obj33"
## [29] "Obj32" "Obj31" "Obj30" "Obj29" "Obj28" "Obj27" "Obj26"
## [36] "Obj25" "Obj24" "Obj23" "Obj22" "Obj21"
##
## [[3]]
##  [1] "Obj1"  "Obj2"  "Obj3"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj13" "Obj14"
## [15] "Obj16" "Obj17" "Obj15" "Obj18" "Obj19" "Obj20" "Obj31"
## [22] "Obj32" "Obj33" "Obj34" "Obj35" "Obj36" "Obj37" "Obj38"
## [29] "Obj39" "Obj40" "Obj21" "Obj22" "Obj23" "Obj24" "Obj25"
## [36] "Obj26" "Obj27" "Obj28" "Obj29" "Obj30"
```

Comment 2: All three `MC` algorithms ranked objects Obj1-Obj10, in that order, in the top-10. `MC2` also ranked the next two groups, Obj11-Obj15 and Obj16-Obj20 in the correct order. However, the remaining objects (Obj21-Obj40) were ranked in the reverse of the expected order. `MC3`, on the other hand, had all the correct ordering except a transposition between the order of objects Obj15 and (Obj16, Obj17). The stationary probabilities can be visualized applying the plotting function (described in the following), which can be used to detect information degradation heuristically.

The stationary probabilities can be plotted by

```
MC.plot(outMC)
```

A plot of the ordered stationary probabilities versus the ranking using the above command can contain useful information regarding the relative rankings of objects. For example, Figure 5 shows that for `MC2`, there was considerable information for ranking objects in the top-20, but there was practically no information ranking any object after that position (all stationary probabilities are the same). This was the reason why the rankings for objects Obj21-Obj40 were completely wrong as we saw in the output. On the other hand, for `MC1`, there was less information for ranking objects beyond the top-10 compared to `MC2`, thus objects Obj11-Obj20 were not ranked correctly. `MC3` appears to be a compromise between `MC1` and `MC2`, leading to a better aggregate ranked list.
Note that the first argument of the `MC.plot` function is the output from the `MC` function, which is mandatory. If the number of ranked objects (second argument) is not specified, the plot will show probabilities for all objects in the union set.

The CEMC algorithm The Order Explicit Algorithm (OEA) in [3] is implemented as the `CEMC` function in `TopKSpace`. Of the three main input arguments in `CEMC`, only the first one (*input*) is required. If argument *space*

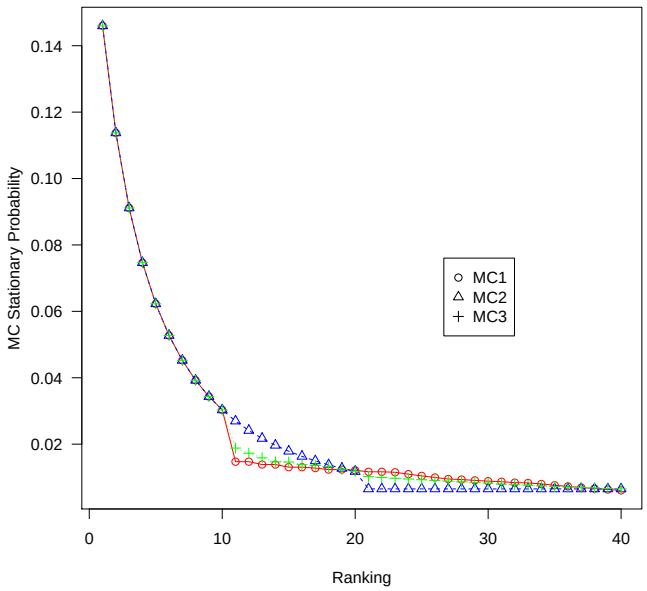


Figure 5: Equilibrium probabilities

is not provided, then all the underlying spaces are assumed to be a common one, and the union of the input top-k lists is supplied as the common space. If argument k is not specified, then the full aggregate ranked-list (all elements in the union) is outputted. The function also has a number of other arguments, all tuning parameters, that can be set to run the OEA more efficiently. The default specified in the `CEMC` function constitutes a set of values that seem to work well for the examples/data that we have analyzed in [3], but it is always a good idea to run OEA with multiple sets of tuning parameters to increase the chance of finding the global maximum; see [3] for some recommendations on how to choose these parameters. In fact, the output of `CEMC` contains the set of tuning parameters used in the current run. If users are unsure of what to use, they are recommended to run `CEMC` without setting any parameter and use the parameter file in the output (*input.par*) to fine-tune the parameters and apply them as input for a more refined analysis. For the convenience of the user, after the parameter values are modified, the file can be directly supplied as an input argument without having to enter the value of each tuning parameter separately. Other than *input.par*, there are two more elements in the output list. One is the aggregate top-k list (`TopK`). The other is the multinomial probability matrix (`ProbMatrix`; each column represents the probability vector of a multinomial distribution and thus sum to 1, albeit with small rounding errors), from which the aggregate top-k list is determined.

```
set.seed(12345)
outCEMC=CEMC(input,space,N=4000,N1=400)
# "space" is explicitly specified; underlying space-dependent
```

```
list(outCEMC$TopK)

## [[1]]
##  [1] "Obj1"  "Obj3"  "Obj2"  "Obj4"  "Obj5"  "Obj6"  "Obj7"
##  [8] "Obj8"  "Obj9"  "Obj10" "Obj11" "Obj12" "Obj16" "Obj13"
## [15] "Obj14" "Obj15" "Obj17" "Obj18" "Obj19" "Obj20" "Obj21"
## [22] "Obj32" "Obj31" "Obj33" "Obj22" "Obj24" "Obj23" "Obj34"
## [29] "Obj25" "Obj35" "Obj36" "Obj37" "Obj26" "Obj38" "Obj39"
## [36] "Obj27" "Obj28" "Obj29" "Obj30" "Obj40"

outCEMC$ProbMatrix[1:5,1:5]

##                  1            2            3            4
## Obj1  1.000000e+00 1.832015e-16 3.100519e-16 3.692052e-16
## Obj10 3.632942e-26 6.619338e-23 3.491686e-20 9.263770e-18
```

```
## Obj11 8.151711e-27 6.532431e-27 1.091877e-26 1.065503e-25
## Obj12 2.233113e-29 5.608852e-27 1.887980e-26 7.906658e-22
## Obj13 1.115825e-26 5.775426e-29 3.430615e-27 1.309585e-22
##                  5
## Obj1  2.580907e-15
## Obj10 5.234333e-15
## Obj11 3.168138e-24
## Obj12 2.111160e-21
## Obj13 3.261699e-25
```

Comment 1: From the submatrix shown above for `CEMC`, it can be seen that item one is ranked no. 1, since the probability vector in the first column for item one is almost 1, but very small for the rest of the items. Similarly, item 2 is ranked no. 2, and so on.

```
outCEMC$input.par

##    k dm  kp    N  N1 extra rho  e1 e2   w b init.m init.w
## 1 40  k 0.5 4000 400     0 0.1 0.1  1 0.5 0      p      0
```

Comment 2: This file contains all the tuning parameter values (defaults) for running `CEMC`. The user may modify this file and use it as the input to the *input.par* argument in the `CEMC` function.

Comparison of performance of the different algorithms In the example given above, we discussed the performances of the different algorithms and assumptions since the underlying truth can be guessed to a large extend since the example was contrived. However, in a real data application, the ground truth is typically completely unknown, therefore, it would be helpful to have some objective criterion to evaluate the relative performances of the various algorithms. To this end, we have implemented two functions based on the modified Kendall distance (MKD) [5] in the `TopKSpace` module. The first function is `Kendall`, which computes the weighted sum of MKD between an aggregate top-k list with the input lists. Of the input arguments, the first two (*input, aggregate*) are required. If argument *space* is not supplied, all input lists are assumed to come from the same underlying space, the union of all elements in the input. If user has prior information on which list is more informative, then that information can be utilized by specifying the weight vector w. The output of `Kendall` is the `MKD`, a single number. However, it would be much more meaningful to compare `MKD`s accross results from different algorithms to assess their relative performances. This can be achieved by invoking the `Kendall.plot` function, which not only provides a vector of `MKD`s but also provides a plot of the `MKD`s for ease of comparison visually. There are also two mandatory input arguments, `input` and `all.aggregates`, where `all.aggregates` is a list comprising aggregate top-k lists from different algorithms to be compared.

The following command will compute the `MKD` for the aggregate list form the Borda algorithm `ARM`.

```
KendallMLists(input,space, outBorda$TopK[,1])

## Modified Kendall Distance
##                 0.5153846

all.aggregates=list(outBorda$TopK[,1],outBorda$TopK[,2],outBorda$TopK[,3],
  outBorda$TopK[,4],outMC$MC1.TopK,outMC$MC2.TopK,outMC$MC3.TopK,outCEMC$TopK)
```

```
Kendall.plot(input,all.aggregates,space,algorithm=c("ARM","MED","GEO","L2N","MC1","MC2","MC3","CEMC"))
```

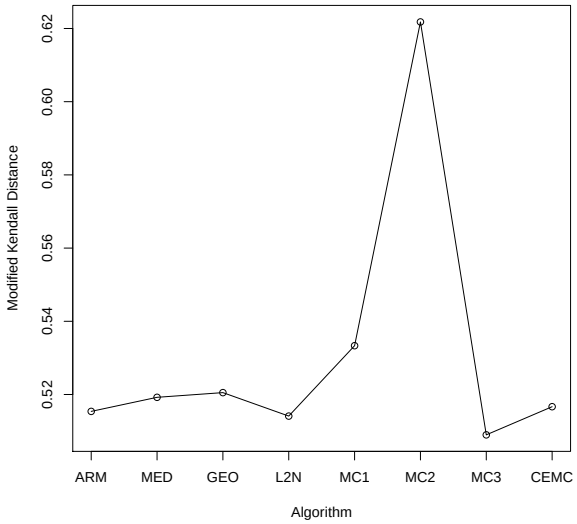


Figure 6: Comparison of the modified Kendall distances across several algorithms

```
## $`Modified Kendall Distance`
##       ARM       MED       GEO       L2N       MC1       MC2
## 0.5153846 0.5192308 0.5205128 0.5141026 0.5333333 0.6217949
##       MC3      CEMC
## 0.5089744 0.5166667
```

Runing the `Kendall` function returns the `MKD` for the aggregate list from the Borda algorithm `ARM`, whereas the `Kendall.plot` function returns a vector of `MKDs`, with the first one matching the output from the `Kendall` function. Figure 6 is the graphical component of the `Kendall.plot` output. As we can see, the range on the y-axis is very small, indicating similar performance of all algorithms, especially those from `Borda`.

2.3 TopKGraphics

2.3.1 Deltaplot

The function `deltaplot` provides an exploratory graph (for examples see Figures 7 and 8), designed to help the user with selecting the distance parameter δ in the `TopKListsGUI` interface. The input for the moderate deviation-based inference procedure is a sequence of I's, taking either zero or one, forming a data stream representing the concordance of the paired ranks of an object o. The data stream depends on some value for the distance δ. The parameter δ is defined by the shift in index positions of a particular object o in one list, say L_i, with respect to the other list, say L_j. This means that we assume concordance (i.e. $I = 1$) for an arbitrary object characterized by rank positions in L_i versus L_j , maximal δ index values apart. For the identification of an appropriate δ in real data analysis, the following strategy is employed: we compute all data streams for $\delta \in [0, 1, 2, \ldots, N-1]$ and order the data stream vectors column-wise according to increasing δ values. In this way, we obtain a $N \times N$ matrix Δ. The ordered sequence of column sums (i.e. the number of 0's) for $\delta \in [0, 1, 2, \ldots, N-1]$) is the information we take advantage of in the so-called *deltaplot*. It represents the reduction of discordance as a function of δ. When all column sums remain zero, complete concordance is attained.

2.3.2 Aggregation map

The aggregation map can only be accessed via the `TopKListsGUI` (see Section 4.2). It can be characterized as follows: We define an index $p = 1, 2, \ldots$ and combine $\ell - 1$ aggregation levels (groupings of truncated lists) in one display: For each group of $\ell - p$ truncated lists down to the smallest group consisting of just one pair of lists, we (i) select an arbitrary reference list L^0 under the condition that it comprises $\max_i(\hat{k}_i)$ items among all pairwise comparisons in the group of rankings, (ii) print the symbols of its $\max_i(\hat{k}_i)$ items vertically from the highest to the lowest rank position, and (iii) add the aggregation information for all remaining $\ell - p$ rankings (pairwise list combinations) in the group, ordered according to descending list length. The aggregation map of the breast cancer dataset can be seen in Figure 14.

2.3.3 Venn diagram

The Venn diagram can only be accessed via the `TopKListsGUI` (see Section 4.2). The Venn diagram, and the respective Venn table, show overlaps of objects in the top-k lists for all analyzed input lists. Asterisk-tagged objects are those from the final aggregate list, produced by `TopKSpace`.

3 Parameter selection

For making inference on pairs of ranked lists, tuning parameters have to be specified. This is for the following reason: For a given set of N objects, arbitrary ranked lists can be constructed by successive permutations. However, this fact does not help in practice when we have to analyze realizations of such lists because they comprise irregularities in terms of position shifts, inverted orderings, missing assignments, etc. As a consequence, a unique top-k list or a complete set of top-conforming objects does not exist. This is the reason why the truncation algorithm (and any other algorithm) in `TopKLists` needs to be controlled by tuning parameters. We therefore adhere to the well established notion of top-k list**s**. By definition, a top-k list consists of k items. The next index value after k is j_0, the point of overlap degradation (k=j_0-1).

There are two main parameters that need to be specified for the truncation algorithm:

1. distance parameter δ
2. pilot sample size ν

Besides the δ and ν parameters, the algorithm uses a constant C which allows us to compensate for poor separability between the informative top parts and the remaining random parts of the input lists (the suggested interval is $0.25 < C < 0.6$). This constant is set to 0.251 in the code (this value is sufficient in most situations).

3.1 δ selection - deltaplot

The choice of the distance parameter δ (which takes the value $0, 1, 2, 3, \ldots$) is crucial for the truncation algorithm. The `deltaplot` of the module `TopKGraphics` was designed to help the user with the selection of δ for any pair of input lists of the same length. A reasonable choice for the distance parameter is associated with a distinct decline in the number of 0's. Generally one should choose a value for δ, where the rate of the deltaplot's decrease begins to slow noticeably (i.e. where the discordance is starting to degrade). When this plot indicates more than one feasible δ-value, preference should be given to the smallest value. Of course, prior information about the ranking mechanisms involved and the nature of the data is also relevant for the selection of δ.

```
deltaplot.dir =  paste0(tempdir(), "/deltaplot")
dir.create(deltaplot.dir, showWarnings = FALSE)
subplot.dir = paste0(tempdir(), "/subplot")
dir.create(subplot.dir, showWarnings = FALSE)
```

```
library(TopKLists)
data(breast)
a=deltaplot(breast, deltas = seq(0,300, by=5), directory=deltaplot.dir)
```

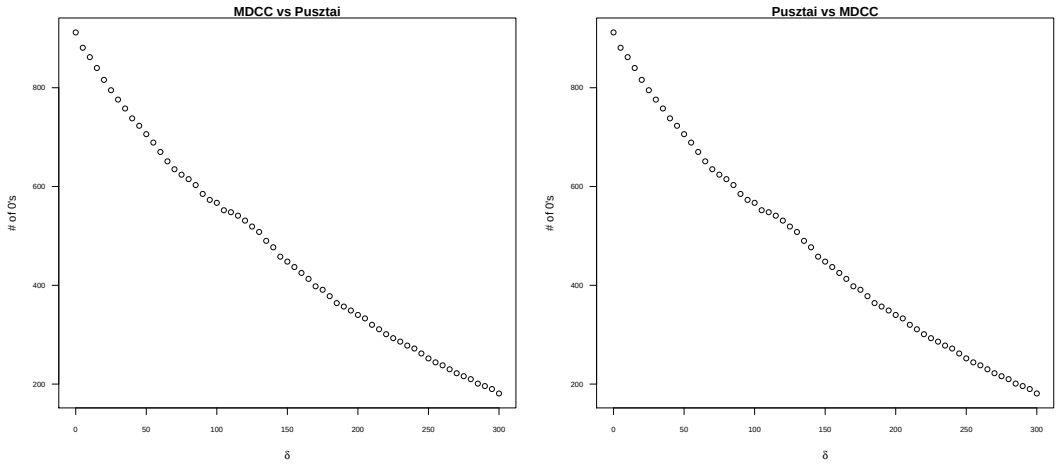


Figure 7: Deltaplot for lists MDCC vs. Pusztai and vice versa

When observing the deltaplot for the first two breast cancer lists (Figure 7), it is not immediately apparent which value for δ should be used. This is because there is no visible point where the `deltaplot`'s decrease is changing its direction.

It can also be observed from Figure 7 that switching the reference list from, for example `Putsztai` to `MDCC` or vice versa, has only little impact on the graph. Therefore, the order of the lists is not relevant. In order to choose a suitable value for δ, we must limit the deltaplot calculation to a smaller subset of the investigated lists. There is no general rule how large this subset should be, but users are encouraged to try a subset of the first 20% - 50% objects. One should search for those points where the decrease suddenly slows down. If there are several such points, the user should consider the smallest one as the suitable δ.

In our breast cancer dataset (of length 917), we took approximately the first 20%, specifically the first 200 objects. This subset can be specified directly in the `deltaplot` function as using the parameter `subset.lists`. Please note that this parameter takes the number of objects (not the percentage) as an input. Additionally, the `subplot` parameter can be used, which magnifies a selected range of small δ-values in a subplot, defined by the `perc.subplot` parameter (the input is a percentage of the main graph, e.g. 50% of the graph is defined as `perc.subplot=50`).

```
a=deltaplot(breast, deltas = 1:50, subset.lists=200, subplot = TRUE,
  perc.subplot=50, directory=subplot.dir)
```

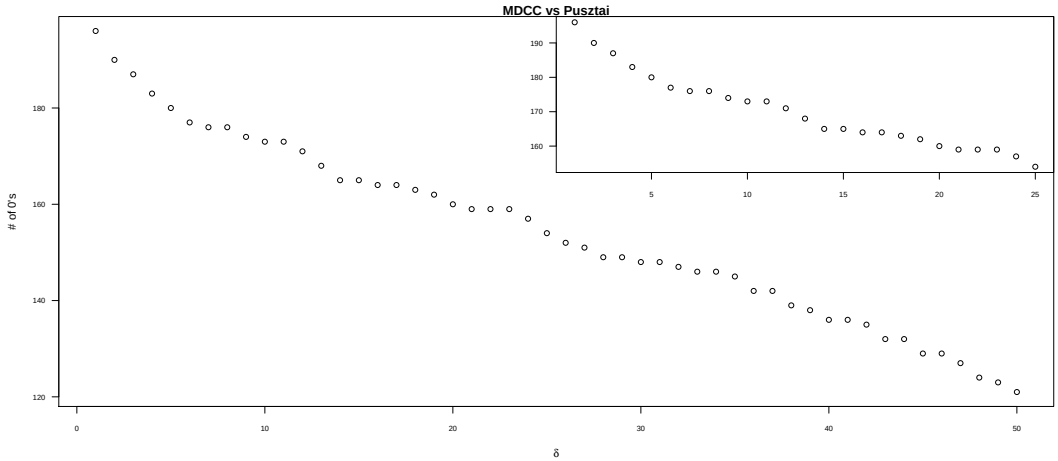


Figure 8: Deltaplot for lists MDCC vs. Pusztai for first 200 objects

In Figure 8, it can be seen that after $\delta = 6$ the decrease of the deltaplot is not as pronounced as for the values $7, 8, 9$ and 10. Therefore, $\delta = 6$ would be the first reasonable choice.

3.2 ν selection

Another important tuning parameter is the pilot sample size ν, which controls the degree of irregularity of the rankings. For this smoothing parameter any positive integer value is admissible. The effect of the choice of the ν parameter on an arbitrary dataset can be seen in section 2.1.2. For most datasets values between 2 and 10 are appropriate. Very large datasets require larger values of ν in the order of tens. For our breast cancer data of length 917, we used $\nu = 10$.

4 Data analysis

The purpose of the `TopKLists` package is to answer the following two main questions:

1. What is the top-k range of highest rank overlap between lists?
2. Which objects in which order form the (top-k) consolidated common list?

The overall procedure leading to these answers is depicted in Figure 10. Red is connected to the search for the index top-k and blue to the aggregation of lists. Both tasks can be targeted independently for arbitrary list lengths, but users should be warned, that the aggregation functions are computationally extensive and might take a significant amount of time. That is why we recommend, for lists longer than about 100, to truncate them at the estimated index max $\hat{k}$ to speed up the aggregation algorithms. As mentioned in the introduction, the user can achieve these results using the console or the GUI.

4.1 Via console

4.1.1 Estimation of the index k

The calculation of the overall truncation index k for pre-specified values of δ and ν (see section 3 about their choice) is performed using the `j0.multi` function. This function takes the input lists, calculates for all pairwise list combinations `Idata` vectors of 0's and 1's (see section 2.1), and then estimates the $\hat{j}_0 = \hat{k} + 1$. Finally, the k is calculated as the maximum of all pairwise $\hat{k}$ values, and outputted as `maxK`. Besides that, the function provides partial results for each pair of lists (under `L`), and the `Idata` vector for each pair of lists (under `Idata`).

```
library(TopKLists)
data(breast)
res = j0.multi(breast, d=6, v=10)
sapply(res, head)

## $maxK
## [1] 14
##
## $L
##      list1    list2  v j0_est  k delta
## 1 TransBig     MDCC 10      9  8     6
## 2 TransBig  Pusztai 10      9  8     6
## 3     MDCC TransBig 10     11 10     6
## 4     MDCC  Pusztai 10     13 12     6
## 5  Pusztai TransBig 10     11 10     6
## 6  Pusztai     MDCC 10     15 14     6
##
## $Idata
##      TransBig_MDCC TransBig_Pusztai MDCC_TransBig
## [1,]             1                1             1
## [2,]             1                1             1
## [3,]             1                1             1
## [4,]             1                1             1
## [5,]             0                0             1
## [6,]             1                1             1
##      MDCC_Pusztai Pusztai_TransBig Pusztai_MDCC
## [1,]            1                1            1
## [2,]            1                1            1
```

```
## [3,]            1                1            1
## [4,]            1                1            1
## [5,]            1                1            1
## [6,]            1                0            1
```

We can see that the estimated `maxK` is 14, i.e. there is a strong concordance between the first 14 objects of the input lists and any concordance beyond the index 14 is most likely due to random effects.

The function `j0.multi` itself uses the functions `prepare.idata` to prepare the `Idata` vector of 0's and 1's, and the function `compute.stream` to calculate the point of degradation $\hat{j}_0 = \hat{k} + 1$ (see the Reference Manual for more information on these functions).

4.1.2 Aggregation of lists

To aggregate multiple lists, here we use the three different aggregation methods (`Borda`, `MC`, and `CEMC`) for comparative purposes. First we truncate the lists to the length of $k = 14$, the estimated index for the most overlapping genes of the three lists.

```
k = res$maxK
TransBig=as.character(breast[1:k,1])
MDCC=as.character(breast[1:k,2])
Pusztai=as.character(breast[1:k,3])
input=list(TransBig,MDCC,Pusztai)
```

Then we have to specify the underlying space, i.e. the set of objects the input lists come from. If space is not specified, all lists are treated as coming from a common space defined by the union of all input lists. The underlying space is defined by the parameter `space`, which should be in the `list` format - one set of objects for each input list. This is because each input list could generally come from a different set of objects (different space). In our case, the set of genes contained in each list is not identical and each list can contain a different set of 14 top genes. Since each of the top-14 gene lists come from the common set of 917 genes, we shall define the space as the set of all the genes present in these 3 truncated lists. In this case there are 22 unique genes (variable `common`).

```
common=unique(unlist(input))
space=list(common,common,common)
```

Finally, we can run the aggregation functions.

```
outBorda=Borda(input,space)
outMC=MC(input,space)
outCEMC=CEMC(input,space,N=2000)
outCEMC$TopK[1:k]

##  [1] "ESR1"     "TBC1D9"   "SCUBE2"   "EVL"      "FBP1"
##  [6] "CIRBP"    "RHOB"     "BTG2"     "FUT8"     "QDPR"
## [11] "IDUA"     "C1orf106" "NAT2"     "GSTM3"
```

The output printed gives the consolidated top-14 genes in `CEMC`-optimized rank order. The `Borda`, `MC` and `CEMC` results can be displayed in matrix form for comparison. As described in detail in section 2.2.2, the `Borda` function comprises four Borda algorithms - arithmetic mean (ARM), median (MED), geometric mean (GEO), and L2-norm(L2N). The `MC` function comprises three Markov chain algorithms - MC1 (spam sensitive), MC2 (majority rule), and MC3 (proportional). The `CEMC` function consist of solely one algorithm, Cross Entropy Monte Carlo.

```
agg=list(ARM=outBorda$TopK[,1],MED=outBorda$TopK[,2],GEO=outBorda$TopK[,3],
  L2N=outBorda$TopK[,4],MC1=outMC$MC1.TopK,
  MC2=outMC$MC2.TopK,MC3=outMC$MC3.TopK,CEMC=outCEMC$TopK)
head(do.call(cbind, agg))
```

```
##       ARM       MED       GEO       L2N       MC1       MC2
## [1,] "ESR1"    "ESR1"    "ESR1"    "ESR1"    "ESR1"    "ESR1"
## [2,] "TBC1D9"  "TBC1D9"  "TBC1D9"  "TBC1D9"  "TBC1D9"  "TBC1D9"
## [3,] "SCUBE2"  "SCUBE2"  "SCUBE2"  "SCUBE2"  "SCUBE2"  "SCUBE2"
## [4,] "EVL"     "EVL"     "EVL"     "EVL"     "EVL"     "EVL"
## [5,] "FBP1"    "FBP1"    "FBP1"    "FBP1"    "FBP1"    "FBP1"
## [6,] "CIRBP"   "CIRBP"   "CIRBP"   "CIRBP"   "CIRBP"   "CIRBP"
##       MC3       CEMC
## [1,] "ESR1"    "ESR1"
## [2,] "TBC1D9"  "TBC1D9"
## [3,] "SCUBE2"  "SCUBE2"
## [4,] "EVL"     "EVL"
## [5,] "FBP1"    "FBP1"
## [6,] "CIRBP"   "CIRBP"
```

Additionally, we can plot the results from the different functions.

```
Kendall.plot(input,agg,space,algorithm=c("ARM","MED","GEO","L2N","MC1","MC2","MC3","CEMC"))
```

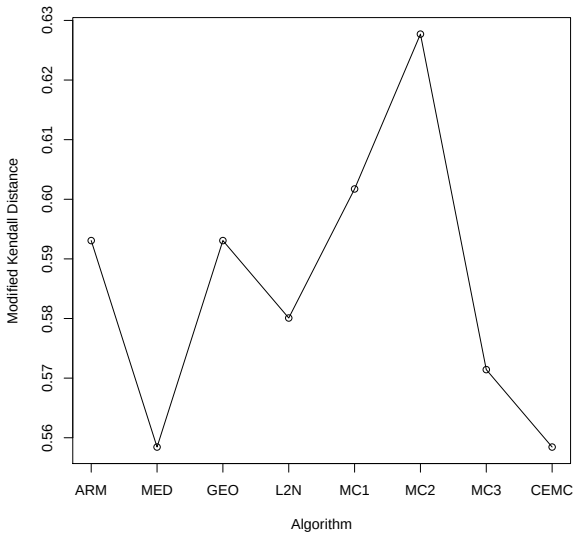


Figure 9: Results from different algorithms for the breast cancer example data

```
## $`Modified Kendall Distance`
##       ARM       MED       GEO       L2N       MC1       MC2
## 0.5930736 0.5584416 0.5930736 0.5800866 0.6017316 0.6277056
##       MC3      CEMC
## 0.5714286 0.5584416
```

Note: The results indicate that Borda with the median function (MED) and the CEMC algorithm perform the best in this example as they lead to the smallest distance between the aggregate and individual lists (Figure 9).

4.2 Via the GUI

The `TopKLists` package comprises a graphical user interface `TopKListsGUI` that offers applied researchers easy and straightforward access to all the functionality of `TopKInference` as well as access to the aggregation technique `CEMC` of `TopKSpace`, and the `aggmap` graphical aggregation tool contained in `TopKGraphics`. Moreover, Venn-diagrams and Venn-tables for the summary of rank aggregation results are accessible. The suite of functions supported by the `TopKListsGUI` facilitates the exploratory analysis of multiple ranked lists. Interactive input facilities (such as a slider for the dynamic specification of the distance δ) allow for the real-time analysis and visualization of results. These results correspond to the distance δ or the pilot sample size ν (note that, for large

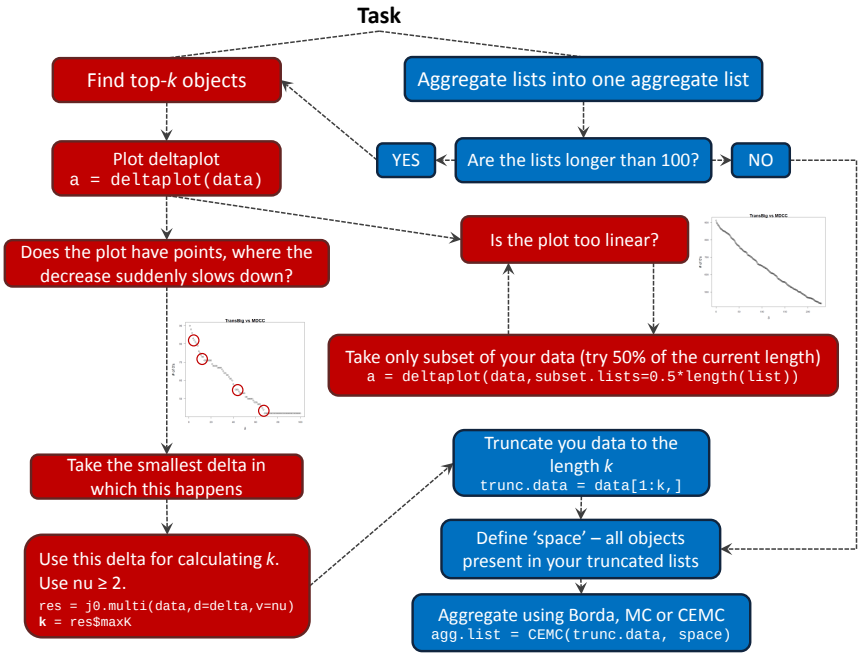


Figure 10: Flowchart

list lengths of N, many numbers of calculations must be performed and longer waiting times should be expected).

To open the GUI using the breast cancer data introduced in section 1.2, run

```
TopKListsGUI(breast)
```

The window shown in Figure11 appears, allowing the user to perform an analysis of two or more lists. The TopKListsGUI window is divided into four panels: arguments, analysis status, delta-slider and results.
In the arguments panel (Figure 12), the user is informed about the number of lists and objects in the dataset. Further, the user can set up variable values for the top-k list computation. The user must select ν (default value is 10) and choose the δ range for which the top-k lists should be calculated (the default is from 0 to 10, in steps of 1). A threshold for the minimum percentage of top lists comprising an object, used for gray-shading in the aggregation map, can also be selected (default is 50). The `TopKSpace` algorithm for the generation of the aggregate list is set by default to `CEMC`. Last but not least, this panel contains the calculate button, which starts the analysis. In the bottom section of `TopKListsGUI` the status of the analysis is shown.
Once the analysis is complete, the delta-slider (Figure 13) allows the user to switch between the results, obtained using different values for δ interactively. The moving bar shows the current value of δ, and the aggregation map updates accordingly. Moving the bar to the left or right allows the user to see results for smaller or larger δ values.

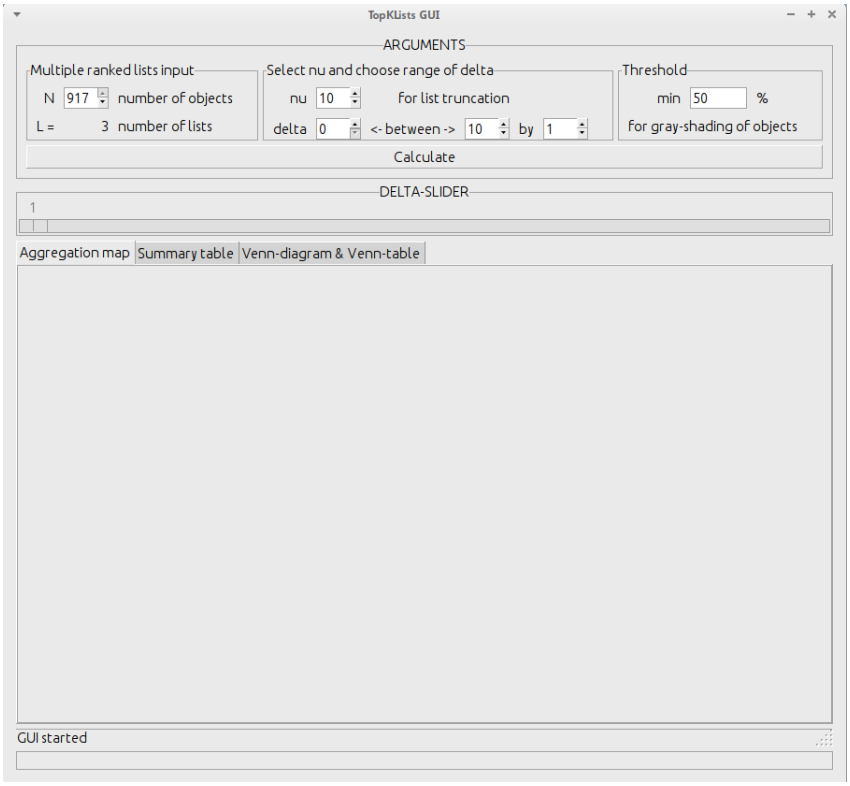


Figure 11: Main window of the `TopKListsGUI`

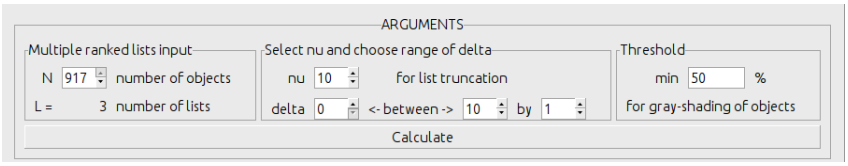


Figure 12: The arguments panel of the `TopKListsGUI` main window

Figure 13: The delta-slider section of the `TopKListsGUI` main window

5 Graphical representation

Once the analysis is complete, the main panel in the GUI serves the visualization of the results for a selected δ value (the position of the delta-slider). The results of the analysis are displayed in three tabs, described below.

5.1 Aggregation map

The aggregation map (Figure 14) shows the top-k list of overlapping genes and their respective distances. It displays aggregation groups, each with a different reference list L^0 (in our case two groups - first with `Pusztai` as the reference list, second with `MDCC` as the reference list). The aggregation information per symbol, item, and group consists of three measures represented by colored triangles and rectangles, respectively, outlined in an array format:

a) The **membership** of an individual item in the top-*k* lists. *Yes* is denoted by the color 'grey' and *no* by the color 'white'.

b) The **distance** d of the rank of an individual item $o \in L^0$ from its position in another list, is denoted by a triangle color scaled from 'red' *identical* to 'yellow' *far distant*. An additional integer value gives the

numerical distance between the item's rank positions, a negative sign means ranked lower, and a positive sign means ranked higher in L with respect to L^0.

c) The rectangular of a symbol takes on the color 'grey' when the **percentage** of $d \leq \delta$ across the columns of a group is above some prespecified threshold, and 'white' otherwise.

As shown in Figure14, the best overlap was found when comparing the MDCC and TransBig lists against the Pusztai list, for $\delta = 6$ (see Section 1 for a description of the example data).

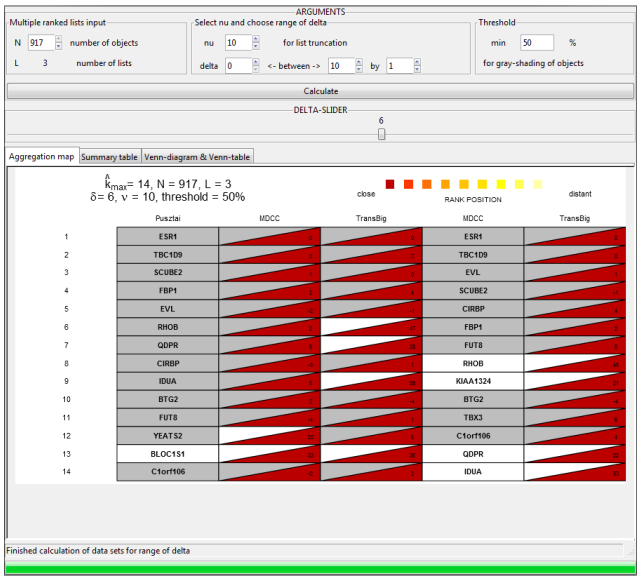
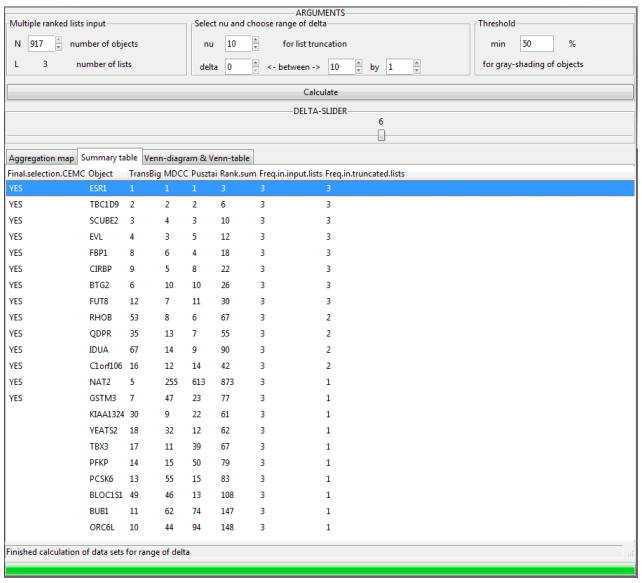


Figure 14: Aggregation map result of the `TopKListsGUI`



Final.selection	CEMC	Object	TransBig	MDCC	Pusztai	Rank.sum	Freq.in.input.lists	Freq.in.truncated.lists
YES		ESR1	1	1	1	3	3	3
YES		TBC1D9	2	2	2	6	3	3
YES		SCUBE2	3	4	3	10	3	3
YES		EVL	4	3	5	12	3	3
YES		FBP1	8	6	4	18	3	3
YES		CIRBP	9	5	8	22	3	3
YES		BTG2	6	10	10	26	3	3
YES		FUT8	12	7	11	30	3	3
YES		RHOB	53	8	6	67	3	2
YES		QDPR	35	13	7	55	3	2
YES		IDUA	67	14	9	90	3	2
YES		C1orf106	16	12	14	42	3	2
YES		NAT2	5	255	613	873	3	1
YES		GSTM3	7	47	23	77	3	1
		KIAA1324	30	9	22	61	3	1
		YEATS2	18	32	12	62	3	1
		TBX3	17	11	39	67	3	1
		PFKP	14	15	50	79	3	1
		PCSK6	13	55	15	83	3	1
		BLOC1S1	49	46	13	108	3	1
		BUB1	11	62	74	147	3	1
		ORC6L	10	44	94	148	3	1

Figure 15: Summary table result of the `TopKListsGUI`

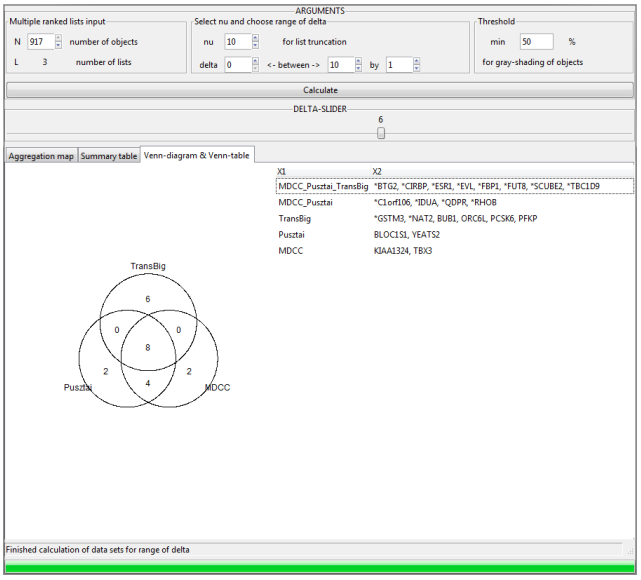


Figure 16: Venn-diagram and Venn-table results of the `TopKListsGUI`

5.2 Summary table

The summary table displays a table of objects present in at least one of the top-k lists, as shown in the aggregation map. The table describes each object name, position of the object in each of the lists, sum of ranks, frequency in the input lists, and frequency in the truncated lists. Please note, that the objects which have assigned the tag 'YES' in the first column are in the top-k set of genes selected by `CEMC` (under default tuning parameters as explained in Section 2.2). The table is ordered in ascending order with regard to the sum of ranks of the objects in the three lists.

5.3 Venn-diagram and Venn-table

The produced Venn-diagram and Venn-table show overlaps of objects in the top-k lists of all input lists in the analysis. The asterisk-tagged objects, in this example the genes, are those that belong to the final aggregate list as returned by the `TopKSpace` module.

6 Summary of breast cancer data results

As can be seen in the previous section, the estimated `maxK` is 14, with 13 genes present in all three top-k lists. Another top-ranked gene (BLOC1S1; rank 13) is only supported by Pusztai. Alltogether, the first 14 genes in the input lists are the most informative ones and should be considered as related to breast cancer. In the summary table (Figure 15), 14 genes are tagged YES, meaning that they belong to the aggregate list calculated by the CEMC method. Most consistent are the ESR1 and TBC1D9 genes, occupying the first two positions in all three lists, as displayed in the aggregation map (with a distance of 0 in both MDCC and TransBig compared to Pusztai) and in the consolidated aggregate list obtained from `CEMC`. The Venn-diagram and Venn-table for the same data provide insight into the overlap characteristics of the top-ranked genes. Platform differences can thus be easily identified.

References

[1] Desmedt C., Piette F., Loi S., Wang Y. et al. (2007). Strong time dependence of the 76-gene prognostic signature for node-negative breast cancer patients in the TRANSBIG multicenter independent validation series. *Clin Cancer Res*, **13**, 3207-3214.

[2] Hall, P. and Schimek, M. G. (2012). Moderate deviation-based inference for random degeneration in paired rank lists. *J. Amer. Statist. Assoc.*, **107**, 661-672.

[3] Lin, S. and Ding, J. (2009). Integration of ranked lists via Cross Entropy Monte Carlo with applications to mRNA and microRNA studies. *Biometrics*, **65**, 9-18.

[4] Lin, S. (2010a). Space oriented rank-based data integration. *Statistical Applications in Genetics and Molecular Biology*, **9**, Article 20.

[5] Lin, S. (2010b). Rank aggregation methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, **2**, 555–570.

[6] Schimek, M. G., Budinská, E., Kugler, K. G., Švendová, V., Ding, J., Lin, S. (2015). TopKLists: a comprehensive R package for statistical inference, stochastic aggregation, and visualization of multiple omics ranked lists. *Statistical Applications in Genetics and Molecular Biology*, **14(3)**: 311-316.

[7] Schimek, M. G., Myšičková, A. and Budinská, E. (2012). An inference and integration approach for the consolidation of ranked lists. *Communications in Statistics - Simulation and Computation*, **41:7**, 1152-1166.

[8] Shi L., Campbell G., Jones W.D., Campagne F. et al. (2010). The MicroArray Quality Control (MAQC)-II study of common practices for the development and validation of microarray-based predictive models. *Nat Biotechnol*, **28**, 827-838.

[9] Tabchy A., Valero V., Vidaurre T., Lluch A. et al. (2010). Evaluation of a 30-gene paclitaxel, fluorouracil, doxorubicin, and cyclophosphamide chemotherapy response predictor in a multicenter randomized trial in breast cancer. *Clin Cancer Res* **16**, 5351-5361.