Vignette: identifying the consensus LDRBO using `consensus_ldrbo`

Philip S. Boonstra

October 31, 2019

Introduction

This vignette presents a step-by-step approach for using the `R` function `consensus_ldrbo` to calculate the consensus list for the patient abstract called 'case A' in the manuscript, which is labeled as 'case23' in the data file.

Ranked versus ordered lists

Pay very close attention the difference between an ordered list and a ranked list. Quoting from Boonstra and Krauss (2019), "Both data types convey equivalent information, and both take the set of all permutations of the v integers [all items that were ranked at least once] as their support. However, whereas a ranked list gives the ranks of the v items, an ordered list permutes the v items themselves based upon their ranking. Specifically, the sth entry of a ranked list is the rank assigned to the item having integer labels (lower numbers indicate higher ranks), and the sth entry of an ordered list is the integer label of the item that is ranked sth (items appearing early in the list are ranked higher)."

Description of algorithm

I use the term 'consensus' below as a synonym for the objective function that is to be maximized (either the median or the mean of the pairwise LDRBO between a single list and each of the observed lists in the data), and the resulting optimized list is called the 'consensus list'. The algorithm works by "growing" the consensus list, starting from the first item, then the second, and so forth. However, it is not just a greedy algorithm in the sense of growing a single list. Potentially many lists are retained at each iteration. The challenge in this approach is that it is impossible to consider all lists unless the number of items is very small. For example, a brute force approach would require evaluating $\sum_{k=1}^{k=n_{\text{items}}} k!$ possible consensus lists, i.e. not feasible for $n_{\text{items}} >\approx 20$. This function uses a 'branch and bound' algorithm to grow lists (the branches) and evaluate their best-case consensus, defined as the consensus that could be achieved after they are fully grown and achieve perfect agreement with all lists on all remaining items to be ranked. Those lists that cannot attain better consensus than what has already been attained, even assuming a best case scenario, can be discarded as clearly suboptimal. However, this is a fairly extreme and simplistic pruning approach, since it is inconceivable except in the most trivial case that any candidate list could attain perfect agreement with *all* lists in the data set. Therefore, empirically I've noticed that most lists are not dropped based upon this check until late in the algorithm, when computing savings are minimal. Therefore, the algorithm further prioritizes lists that cannot be dropped based on the relative difference between (i) the current consensus that the list has achieved and (ii) maximum possible consensus that the list can achieve assuming perfect agreement. More technically, if y is the list that is currently the best list under consideration, then for an arbitrary list x I calculate

criterion: $\max(0, [x(max) - y(obs)]/[x(max) - x(obs)])$

where $x(max)$ is the theoretical best possible consensus that x can achieve after adding more items given its current level of consensus, $x(obs)$ is the current consensus that x achieves, and $y(obs)$ is consensus of list that is currently best. Lists that need to drastically exceed the current level of consensus already achieved by list y (the numerator), relative to what they have already achieved (the denominator) are not likely candidates for ultimately being the consensus list and therefore prioritized lower. Larger values of this criterion are better, and list y will always evaluate to 1, the best possible value. Additionally, all lists for which this criterion evalutes to 0 can be automatically dropped, since this implies that $x(max) \leq y(obs)$, i.e. whatever x can become, even in the best case, cannot be preferred to list y in its current form

Example: Case A

Read in data, functions

```
library(tidyverse);
source("gather_data.R");
source("functions_ldrbo.R");
```

Immediately below are the data formulated as ranked lists. Note that each column indicates a problem, and so an NA means that a physician (row) did not include that problem in his/her list. The number of columns is the number of unique items ranked. If you are interested in how physicians ranked a specific problem, then a set of ranked lists is useful because you can quickly scan down a column to see where physicians put it on their list.

```
as_tibble(case23_ranked);
```

```
## # A tibble: 32 x 28
##    PNEUMONIA ANEMIA HYPOPHOSPHATEMIA SPLENOMEGALY `HISTORY OF SMO~
##        <int>  <int>            <int>        <int>            <int>
##  1         1      2                3            4                5
##  2         1     10               NA            9               NA
##  3         1      3               NA            4               NA
##  4         1      3               NA           NA                5
##  5         1      2               NA            3                8
##  6         1      7               10           NA               NA
##  7         1     NA               NA            3               NA
##  8         4      6               NA           NA               NA
##  9         1      3               NA            4               NA
## 10         1      4               NA           NA               NA
## # ... with 22 more rows, and 23 more variables: `DIABETES MELLITUS` <int>,
## #   OSTEOARTHRITIS <int>, `DEPRESSION WITH ANXIETY` <int>, `POST
## #   MENOPAUSAL ON HRT` <int>, `LOWER EXTREMITY EDEMA` <int>, `CHEST
## #   PAIN` <int>, `SYSTOLIC MURMUR` <int>, `RENAL FAILURE` <int>,
## #   `CONGESTIVE HEART FAILURE` <int>, PANCYTOPENIA <int>, FEVER <int>,
## #   TACHYCARDIA <int>, `IRON DEFICIENCY` <int>, THROMBOCYTOPENIA <int>,
## #   HYPOXEMIA <int>, `SHORT OF BREATH` <int>, `PULMONARY EDEMA` <int>,
## #   `PULMONARY EMBOLISM` <int>, `HIGH HAPTOGLOBIN` <int>,
## #   DEPRESSION <int>, ANXIETY <int>, HYPOXIA <int>, HYPOALBUMINEMIA <int>
```

In contrast, below is the same data formulated as a matrix of ordered lists (for brevity, I only plot the first five lists). Each row is still a physician, but now each column is a rank. Here, an NA means that the list in that row has already ended, and it is just a filler to make the matrix rectangular. No NA should precede a non-NA. The number of columns is (or should be, rather) the length of the longest list. I say 'should be' because you can always trivially add a column of NAs; but don't do this. In other words, the last column in your matrix of ordered lists should contain at least one row with a non-NA. If you are interested in a specific rank, then a set of ordered lists is useful because you can quickly scan down a column to see the items that are most frequently in that rank. **To use the functions in this repository, your data must be formualted as ordered lists**

```
case23_ordered[1:5,];
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    2    3    4    5    6    7    8    9    10    NA    NA    NA
## [2,]    1   11   12   13    6    7    9    8    4     2    NA    NA    NA
## [3,]    1   14    2    4   13    6    8    7   NA    NA    NA    NA    NA
## [4,]    1   15    2    6    5    8    7   NA   NA    NA    NA    NA    NA
## [5,]    1    2    4   12    6    9    8    5    7    NA    NA    NA    NA
##      [,14]
## [1,]    NA
## [2,]    NA
## [3,]    NA
## [4,]    NA
## [5,]    NA
```

Inspect pairwise LDRBOs

First, I calculate the matrix of pairwise LDRBO values for the 32 physicians.

```
# Calculate and store the results
case23_pairwise_ldrbo <-
  ldrbo(dat_new = case23_ordered,
        psi = 1,
        verbose_results = FALSE);

# This is the vector of 32 LDRBO values comparing the list from physician 1
# to the lists from physicians 1, ..., 32. By definition, the LDRBO of an
# identical list, i.e. comparing physician 1's list to itself, must be 1

case23_pairwise_ldrbo[,1];
```

```
##  [1] 1.000 0.491 0.677 0.666 0.769 0.405 0.477 0.218 0.610 0.402 0.784
## [12] 0.694 0.679 0.460 0.344 0.460 0.639 0.637 0.508 0.741 0.219 0.391
## [23] 0.305 0.460 0.470 0.229 0.628 0.634 0.687 0.668 0.439 0.525
```

Calculate the consensus LDRBO

Now I use the function `consensus_ldrbo` to calculate the consensus list that has the maximum median pairwise LDRBO across all lists in our data. At a minimum, I only need to provide the data and the choice of ψ.

```
begin = Sys.time();
case23_consensus_ldrbo <-
  consensus_ldrbo(dat = case23_ordered,
                  psi = 1);
# total running time required
(default_running_time <- difftime(Sys.time(), begin, units = "secs"));
```

```
## Time difference of 13.2 secs
```

The function returns a list of various helpful things, most importantly of which is the `consensus_list`, which is given as an ordered list

```
case23_consensus_ldrbo$consensus_list;
```

```
## [1]  1  6  2  4  8  7 13 27
```

To see what actual problems these correspond to, recall that these are the column indices of the data when *formulated as ranked lists*, i.e.

```
colnames(case23_ranked)[case23_consensus_ldrbo$consensus_list];
```

```
## [1] "PNEUMONIA"                "DIABETES MELLITUS"
## [3] "ANEMIA"                   "SPLENOMEGALY"
## [5] "DEPRESSION WITH ANXIETY" "OSTEOARTHRITIS"
## [7] "RENAL FAILURE"            "HYPOXIA"
```

This should be equivalent to the LDRBO column in Table 4 in Boonstra and Krauss (2019). Type

```
case23_consensus_ldrbo$consensus_total_rbo
```

```
## [1] 0.683
```

to see what the maximized median pairwise LDRBO actually was. You should not be able to find a list that has a larger median pairwise LDRBO with these 32 ordered lists; however, because this algorithm is not guaranteed to find the global optimum, it is possible that such a list may exist. You can use the `ldrbo` function to re-calculate pairwise LDRBO between this consensus list and the data using the following:

```
ldrbo(dat_new = matrix(case23_consensus_ldrbo$consensus_list,nrow = 1),
      psi = 1,
      dat_ref = case23_ordered,
      verbose_results = FALSE);
```

```
##       [,1]  [,2]  [,3] [,4] [,5]  [,6] [,7]  [,8]  [,9] [,10] [,11] [,12]
## [1,] 0.691 0.555 0.739 0.69 0.71 0.397 0.53 0.229 0.682 0.484 0.758 0.736
##      [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23]
## [1,]  0.71 0.688 0.534 0.688 0.685 0.749 0.486 0.739 0.255 0.634 0.436
##      [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32]
## [1,] 0.641 0.549 0.405 0.629 0.699 0.739 0.684 0.688 0.563
```

(I had to force the consensus list into a matrix with 1 row). If you take the median of these values, it should be exactly equal to the value of `case23_consensus_ldrbo$consensus_total_rbo` above, namely 0.683.

Faster, greedier versions

You will notice the algorithm above took about 13.2 seconds to finish. If you're willing to sacrifice some potential optimality in your solution, you can make the algorithm run faster by specifying some of the options in the `consensus_ldrbo` function. Specifically, you can provide your own values for the arguments `window_seq` and/or `look_beyond_init`, `look_beyond_final`.

If not provided, `ldrbo_consensus` calculates these automatically. I can see what these values were in our initial run using the following:

```
case23_consensus_ldrbo$control$window_seq;
```

```
##  [1]  5  6  7  8  9 10 11 11 12 13 14 15 16 15 14 13 12 11 10  9  8  7  6
## [24]  5  4  3  2  1
```

```
case23_consensus_ldrbo$control$look_beyond;
```

```
##  [1]  Inf 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
## [15] 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
```

The kth element of `window_seq` tells me how many of the next most promising items were considered for the next item in the consensus list, where "promising" is defined as the initial prioritization given by `case23_consensus_ldrbo$control$initial_order` (briefly, the function looks at how frequently each item was ranked, and where, and assigns points based upon these features. More points means more promsing). And the kth element of `look_beyond` tells me the largest number of candidate lists that may be retained at the end of this step.

For example, at step 1, the first 5 items in `initial_order` were evaluated. In the first step only, all lists are always kept. Then, at step 2, each list currently in the set of candidates splits into 6 new lists, with each new list containing the most promising items from `initial_order` that aren't already in this particular candidate list. At step 2, then, there will be `window_seq[1]` $\times$ `window_seq[2]`, or 30, candidate lists. Since this is less then the value of `look_beyond[2]`, I can keep all of them for the time being and proceed to step 3 and so on.

Here I set `window_seq` so that the next most promising three items are only ever considered:

```
n_items = ncol(case23_ranked);
begin = Sys.time();
case23_consensus_ldrbo_mod1 <-
  consensus_ldrbo(dat = case23_ordered,
                  psi = 1,
                  window_seq = rep(3, n_items));
```

```r
# total running time required
(mod1_running_time <- difftime(Sys.time(), begin, units = "secs"));
```

```
## Time difference of 1.14 secs
```

```r
case23_consensus_ldrbo_mod1$consensus_list;
```

```
## [1]  1  2  6  8  4 12
```

```r
case23_consensus_ldrbo_mod1$consensus_total_rbo;
```

```
## [1] 0.669
```

As you can see, the algorithm ran substantially faster but at a cost of finding a slightly suboptimal consensus problem list (with a maximized median pairwise LDRBO of 0.669 versus what I found previously, which was 0.683)

We can also make the algorithm run faster by changing the values of `look_beyond`. The function will take your inputs for `look_beyond_init` and `look_beyond_final` and create `look_beyond` as an equally spaced sequence of integers:

```r
# This is a snippet of code from the function `consensus_ldrbo`
look_beyond = c(0, round(seq(from = look_beyond_init,
                             to = look_beyond_final,
                             length = max_size - 1)));
```

The default values of `look_beyond_init` and `look_beyond_final` are both 1000. If you decrease either of these numbers, the algorithm will keep fewer candidates at each iteration and therefore require fewer function evaluations.

```r
begin = Sys.time();
case23_consensus_ldrbo_mod2 <-
  consensus_ldrbo(dat = case23_ordered,
                  psi = 1,
                  look_beyond_init = 50,
                  look_beyond_final = 50);
# total running time required
(mod2_running_time <- difftime(Sys.time(), begin, units = "secs"));
```

```
## Time difference of 5.82 secs
```

```r
case23_consensus_ldrbo_mod2$consensus_list;
```

```
## [1]  1  6  2  4  8  7 13 27
```

```r
case23_consensus_ldrbo_mod2$consensus_total_rbo;
```

```
## [1] 0.683
```

In this case, I actually identified the same consensus list as was initially found but in about 7.3 fewer seconds.

Fin