Supplementary Material

Anonymous Author(s)
Affiliation
Address
email

1 Proof of Proposition 1

With the notations used in the article, one has

$$\begin{aligned}
R_{LpO} &= \binom{n}{p}^{-1} \sum_{e} \sum_{i \in \bar{e}} \mathbb{I}_{\{f^e(x_i) \neq y_i\}} \\
&= \sum_{i=1}^{n} \binom{n}{p}^{-1} \sum_{e} \mathbb{I}_{\{f^e(x_i) \neq y_i\}} \mathbb{I}_{\{i \in \bar{e}\}} \\
&= \sum_{i=1}^{n} \sum_{e} \mathbb{I}_{\{f^e(x_i) \neq y_i\} \bigcap \{i \in \bar{e}\}} P(E = e) \\
&= \sum_{i=1}^{n} P\left(\{f^E(x_i) \neq y_i\} \bigcap \{i \in \bar{E}\}\right) \\
&= \sum_{i=1}^{n} P\left(f^E(x_i) \neq y_i | i \in \bar{E}\right) P\left(i \in \bar{E}\right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} P\left(f^E(x_i) \neq y_i | i \in \bar{E},\ V_k^i = j\right) P\left(V_k^i = j | i \in \bar{E}\right) P\left(i \in \bar{E}\right) \\
&= \sum_{i=1}^{n} P\left(i \in \bar{E}\right) \sum_{j=k}^{k+p-1} P\left(V_k^i = j | i \in \bar{E}\right) P\left(f^E(x_i) \neq y_i | i \in \bar{E},\ V_k^i = j\right) \quad , \qquad (1)
\end{aligned}$$

which achieves the proof.

2 Algorithms for WkNN Leave-p-out

2.1 Algorithms

Positive weights First, the following problem is considered: assuming that m objects have positive values $w_1 \leq ... \leq w_m$, how many combinations (without replacement) of k of these objects among m lead to a total value higher than s, with $s > 0$? Denote $W = (w_1, ..., w_m)$, and $N(W, s, k)$ the number of combinations for which the condition is fulfilled, and $I(W, s)$ the breakpoint index of W. The breakpoint index is the smallest j such that $\sum_{i \leq j} w_i \geq S$ (if for all j $\sum_{i \leq j} w_i < S$ then $I(W, s) = m + 1$ by convention).

There are several convenient settings where $N(W, s, k)$ can be computed:
- if $k = 1$, then $N(W, s, k) = m - \max\{j / w_j < s\}$,
- if W is of length k, then $N(W, s, k) = 0$ or 1,
- if $I(W, s) \leq k$ then $N(W, s, k) = \binom{m}{k}$,

- if $I(W,s) = m+1$ then $N(W,s,k) = 0$.

Based on these remarks, the proposed algorithm is:

Require: W, s, k
  $L \leftarrow$ length(W)
  $BI \leftarrow$ breakpoint_index(W,s,k)
  $BoolCond \leftarrow$ check_conv_settings(W,s,k,L,BI)
  if $BoolCond = 1$ **then**
    $NumbComb \leftarrow$ compute_numb_comb(W,s,k,L)
  else
    $NumbComb$ = 0
    for $i = BI$ to L **do**
      $NumbComb \leftarrow NumbComb$ + Pos_Weights($W[1:i-1], s-W[i], K-1$)
    end for
  end if
  return $NumbComb$

In practice, this algorithm is faster than the naive algorithm based on recursive programming only (i.e. where the breakpoint index is not computed).

Positive and negative weights We now assume that m_0 objects have negative values $w_1^0 \leq ... \leq w_{m_0}^0$, and m_1 objects have positive values $w_1^1 \leq ... \leq w_{m_1}^1$, and we wonder how many combinations (without replacement) of k of objects among $m_0 + m_1$ lead to a total value higher than s. We note $W_i = (w_1^i, ..., w_{m_i}^i)$ for $i = 0, 1$, $W = (W_0, W_1)$, and denote $N(W_0, W_1, s, k)$ the number of combinations for which the condition is satisfied.

The convenient settings where $N(W_0, W_1, s, k)$ can be computed are the following ones:
- if $k = 1$, then $N(W_0, W_1, s, k) = m - \max\{j \mid w_j < s, \ \ w_j \in W\}$,
- if W is of length k, then $N(W_0, W_1, s, k) = 0$ or 1,

Besides, if either W_0 or W_1 is empty we can use algorithm 2.1 proposed in the previous paragraph. The new algorithm is then:

Require: W_0, W_1, s, k
  $L_1 \leftarrow$ length(W_1)
  $L_2 \leftarrow$ length(W_2)
  $BoolCond \leftarrow$ check_conv_settings(W_1,W_2,s,k,L_1,L_2)
  if $BoolCond = 1$ **then**
    $NumbComb \leftarrow$ compute_numb_comb(W_1,W_2,s,k,L_1,L_2)
  else if is_empty(W_0) **then**
    $NumbComb \leftarrow$ Pos_Weights(W_1, s, K)
  else if is_empty(W_1) **then**
    $NumbComb \leftarrow \binom{k}{m_1}$ - Pos_Weights($-W_0, -s, K$)
  else
    $NumbComb \leftarrow$ PosNeg_Weights($W_0, W_1[1:L_1-1], K-1, s-w_{L_1}^1$)
      + PosNeg_Weights($W_0, W_1[1:L_1-1], K, s$)
  end if
  return $NumbComb$

Notice that the recursive call of the algorithm can be refined by reducing either W_0 or W_1 (depending on which one has the smallest number of items) instead of W_1 only. In this case, the "worst" cases are the one where W_0 and W_1 are of equal size, i.e. intuitively cases where the noise level is high.

2.2 Computational time

Table 2.2 provides the computation time of the weighted procedure, on a sample of 500 observations. The complete distance matrix between observations is calculated beforehand. For each observation, the label is drawn in a Bernoulli distribution $\mathcal{B}(q)$, results are presented for $q = 0.1$ and 0.5. Weights in the majority voting rule are all equal to 1. We observe that the computational times highly depends on the noise level q. As a comparison, for $k = 9$ and $p = 25$, the exact LpO procedure is run within a second.

3 Influence of p on k_p, n fixed

k/m	5	10	15	20	25
3	0.55	1.21	2.16	3.10	4.52
5	0.93	3.53	9.41	19.97	38.79
7	1.35	8.62	42.75	163.67	506.24
9	1.90	22.98	229.55	1484.52	7247.31

$q = 0.1$

k/m	5	10	15	20	25
3	0.65	1.86	3.32	5.34	7.78
5	1.95	11.29	35.80	90.75	192.18
7	3.76	44.60	281.77	1162.73	3822.56
9	6.95	159.37	1774.14	12225.48	57212.10

$q = 0.5$

	$1 < p < 10$	$11 < p < 30$	$40 < p < 80$	$p > 80$	Test
k	21	19	17-15	13-9	17

Table 1: Choice of parameter k by LpO for different values of p, or by test sample, when $q = 0.3$.