Confidentiality for Cox model

The WebDISCO method for estimating a horizontally partitioned Cox model does not ensure data confidentiality because predictors can be retrieved by the central server. This occurs due to sending data for every event time. When only one event happens during a given period, only the predictors of that individual vary, making it possible to identify them.

To address this issue, it is important to ensure that there is never a single event occurring at a single event time. Therefore, data can be divided into intervals where many patients are grouped. Various methods are available for this purpose.

Averaging

The data is ordered by time, and values are grouped and given a new time, which is the average of all values of time.

Example.

For the example, we assume we want to group data into groups of 2.

time	p_1	p_2
2	43	0
4	24	0
5	41	1
6	37	1
9	53	0
11	33	1
12	39	1
17	45	0

Before

time	p_1	p_2
3	43	0
3	24	0
5.5	41	1
5.5	37	1
10	53	0
10	33	1
14.5	39	1
14.5	45	0

After

Where p_1 and p_2 are predictors.

For this method, no communication is required between sites. All is computed locally in only one iteration.

Despite the simplicity of this method, there is a significant issue. The intervals are not consistent across all sites, which can lead to values from different sites switching positions. Specifically, if the same value appears in both sites but is rounded up in one site and rounded down in the other, the positional information of these values is lost. Since the Cox model relies on the order of events rather than the exact timing, this discrepancy can introduce errors.

Uniform Intervals (with cutoff)

The period of the study is split into uniform intervals that contain at least x subjects per interval. The last few values are excluded when choosing the interval size, as they can be spread far apart.

Example

Since the last few values in a survival analysis distribution can be spread far apart, we determine the size of the intervals by excluding a percentage of the last values to avoid having excessively large intervals. In this example, we choose to exclude 15% of the last values. As a result, the last value, 17, is excluded when choosing the interval size.

To ensure there are at least 2 values in every interval, The intervals must be larger than the largest difference between any two-time values separated by one other value. In this case, the largest difference is calculated as $11 - 6 + 1 = 6$. Therefore, the intervals should be of size 6.

time	p_1	p_2
2	43	0
4	24	0
5	41	1
6	37	1
9	53	0
11	33	1
12	39	1
17	45	0

Before

time	p_1	p_2
1	43	0
1	24	0
1	41	1
1	37	1
2	53	0
2	33	1
2	39	1
2	45	0

After

The intervals are: [2, 8[, [8, 14[

The excluded values, here 17, will be added back to the last interval.

This method requires communication between sites and a global server to ensure all sites have the same intervals. The communication process is as follows:

1. Each site sends its cutoff value to the global server according to the percentage of excluded values determined by the global server. The cutoff value is the last time value included after excluding the specified percentage of values.
2. The global server selects the lowest cutoff value received from all sites and sends it back to each site.
3. Knowing the start point of the first interval and the cutoff value, each site calculates the minimum possible size for the intervals with the method presented earlier, and then sends this minimum interval size to the global server.
4. The global server selects the largest minimum interval size from all sites, ensuring that each site has enough values in each interval, and sends it back to the sites.
5. With this interval size, each site can aggregate data locally.

By following this method, all sites are ensured to have the same intervals.

Non-uniform Intervals

The period of the study is split into non-uniform intervals. These intervals are the smallest possible size that contains x subjects.

Example

Each interval should contain the minimum number of values possible. Ideally, each interval will contain exactly 2 values.

time	p_1	p_2
2	43	0
4	24	0
5	41	1
6	37	1
9	53	0
11	33	1
12	39	1
17	45	0

Before

time	p_1	p_2
1	43	0
1	24	0
2	41	1
2	37	1
3	53	0
3	33	1
4	39	1
4	45	0

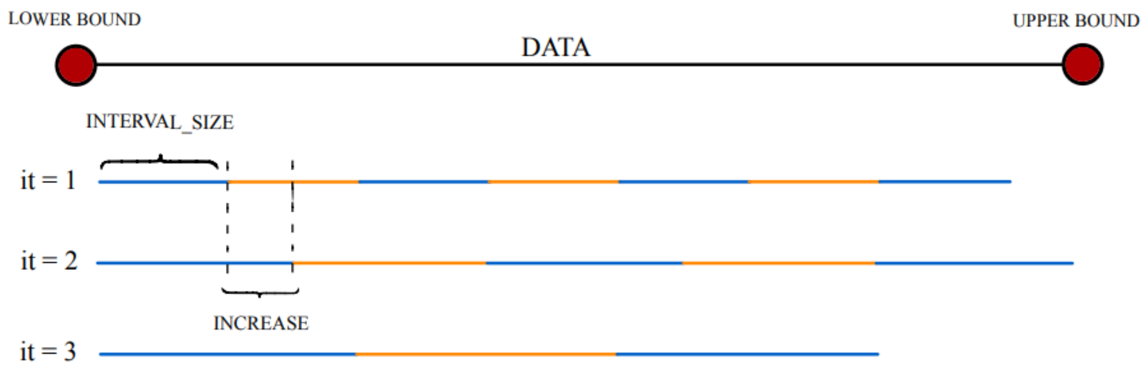
After

Intervals: [2, 5[, [5, 9[, [9, 12[, [12, ∞[.

However, all the sites must agree on these intervals (see next section).

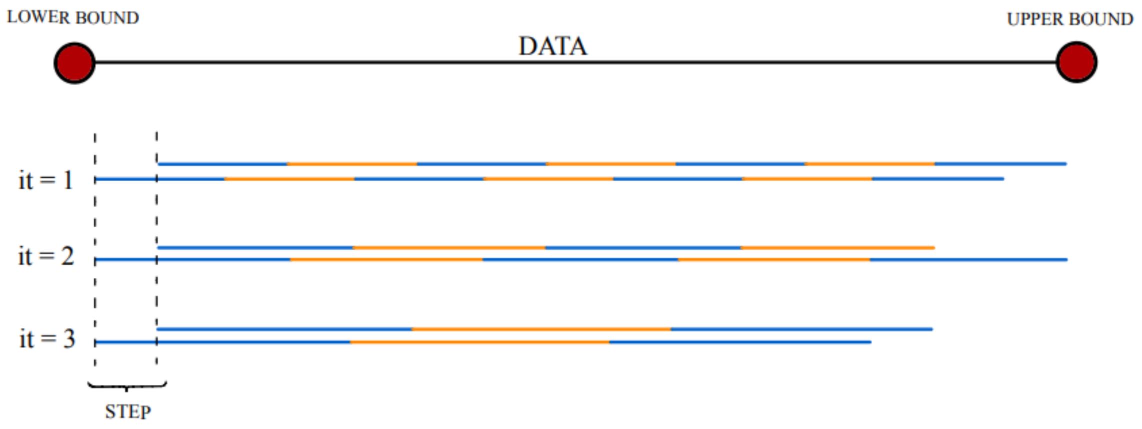
To minimize the amount of communication between sites, an algorithm was implemented that requires only one back and forth with the central server.

First, the local data is split into intervals:

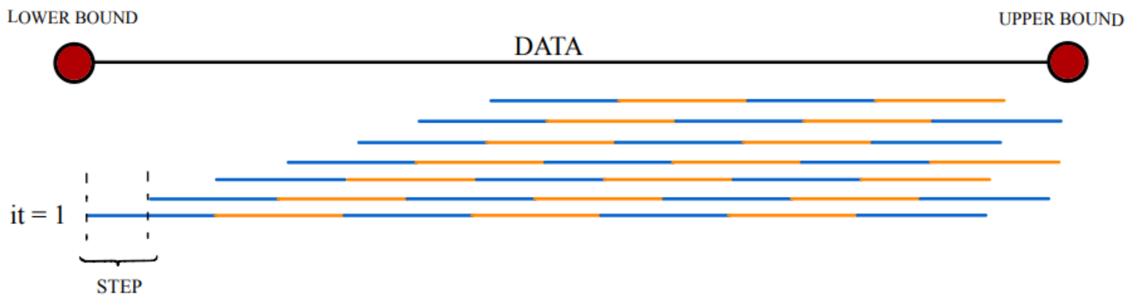


Where *lower bound* is the smallest value, *upper bound* is the biggest value (or the length of the study), *interval size* is the initial size of the interval, and *increase* is the amount that the interval increases every iteration.

To find the smallest interval size possible, it can be interesting to have overlapping intervals:



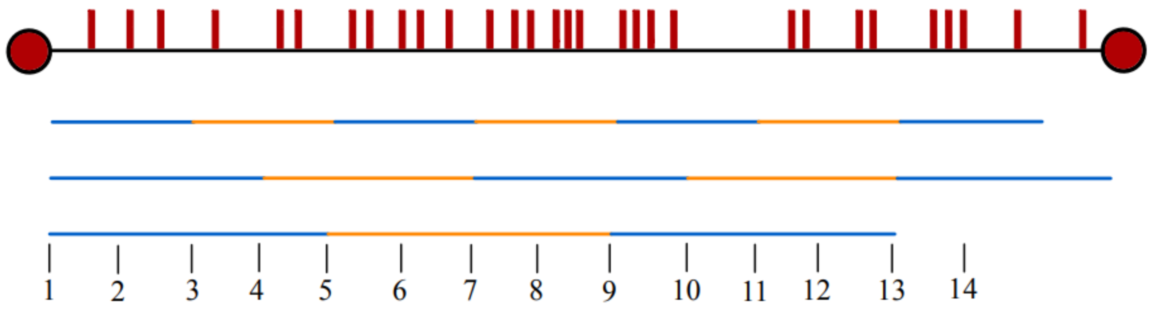
For the first iteration:



The value *step* determines the shift between the start of the intervals. In the first example, *step* was equal to *interval size*. To make sure that all values are contained within intervals of decent size, *step* should always be equal or smaller than the initial *interval size*, or values may fall between two intervals. The best practice, when possible, is for *interval size* and *step* to be equal to the smallest difference between two values in the dataset.

Once the intervals parameters have been determined and the intervals have been computed, the number of values in each interval is checked. If the interval contains more than the minimum number of values necessary, the value 1 is put at associated position in the binary outcome matrix.

For this simple example:



If we want at least 5 values in each interval, the binary output matrix would be:

		Interval start position													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Interval length	**2**	0	0	0	0	1	0	1	0	0	0	0	0	0	0
	3	0	0	0	1	0	0	1	0	0	0	0	0	1	0
	4	1	0	0	0	1	0	0	0	1	0	0	0	0	0

For example, consider intervals that start at position 1. Only an interval of size 4 contains at least 5 values. The interval of size 2 contains only 3 values, and the interval of size 3 contains 4 values.

Another example is at position 5. Here, the interval of size 2 contains 5 values, and the interval of size 4 contains 11 values. No interval of size 3 starts at position 5, which explains the value of 0.

And finally, the interval size can be chosen, by selecting the smallest interval size possible.

		Interval start position													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Interval length	**2**	0	0	0	0	1	0	1	0	0	0	0	0	0	0
	3	0	0	0	1	0	0	1	0	0	0	0	0	1	0
	4	1	0	0	0	1	0	0	0	1	0	0	0	0	0

Each site generates their own binary output matrix. The central server will sum all these matrices. If there are 3 sites, the global output matrix will look something like this:

		Interval start position													
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Interval length	**2**	0	0	1	0	3	0	3	0	2	0	1	0	1	0
	3	1	1	2	1	3	1	2	0	1	0	0	0	3	0
	4	3	0	0	0	3	0	0	0	3	0	0	0	0	0

Instead of selecting the smallest interval size where there is a 1, the interval size selected should be the one where the value is equal to the number of sites (3). This guarantees the interval size contains enough values in all the sites.

The algorithm used for interval size selection is this one:

At interval position 1: interval size 4 is selected. The next position to check is $1 + 4 = 5$.

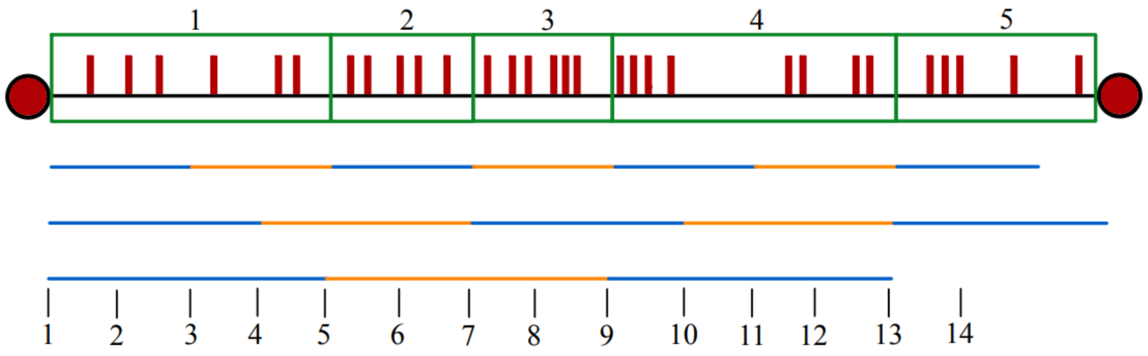
At interval position 5: interval size 2 is selected. The next position to check is $5 + 2 = 7$.

At interval position 7: interval size 2 is selected. The next position to check is $7 + 2 = 9$.

At interval position 9: interval size 4 is selected. The next position to check is $9 + 4 = 13$.

At interval position 13: interval size 3 is selected. It is the last interval. All remaining values after this interval will be added to this one.

For this example, this is what the algorithm would give:


1
2
3
4
5
1
2
3
4
5
6
7
8
9
10
11
12
13
14