RCASPAR: A package for survival time analysis using high-dimensional data

Douaa AS Mugahid

April 11, 2014

1 Introduction

RCASPAR allows the utilization of high-dimensional biological data as covariates for survival time prediction based on the step-wise Cox model. The data is usually that obtained from microarray measurements, but it can be any sort of data thought reasonable for this purpose by the user. Preprocessing must be done by the user, and the final values needed for use in the program are what should be provided. Figure 1 presents a step-by-step overview of how to move through the different functions of the algorithm, there is a wrapper that combines many of the functions to return the final predicted survival times based on the input given by the user.

2 Arranging the input data for *RCASPAR*

The input data for R-CASPAR is basically the gene-expression/aCGH/protein microarray, etc... values to be used as covariates in survival time prediction. The data should be:

1. pre-processed and the provided values ready for use as such.
2. All subjects for which the data is available should have the same number of covariates.
3. Missing covariates for some patients should be included as zeros.

The final form in which the data for all covariates for all patients is to be submitted is as a matrix, in which the rows correspond to the patients and the columns to the covariates (ex: genes) as follows:

Patient	coV.(1)	coV.(2)	coV.(3)	...	coV.(i)
1	x(1,1)	x(1,2)	x(1,3)	x(1,...)	x(1,i)
2	x(2,1)	x(2,2)	x(2,3)	x(2,...)	x(2,i)
...	x(...,1)	x(...,2)	x(...,3)	x(...,...)	x(...,i)
(j)	x(j,1)	x(j,2)	x(j,3)	x(j,...)	x(j,i)

Figure 1: Figure depicting how gene-expression or aCGH data are to be entered for a patient set

Survival information, i.e.: survival times in years, months or days (whatever the user prefers) and censorship status as 0 for an uncensored patient, and 1 for a censored patient is provided in a data frame as follows with at least the following columns, with the exact column names, though extra columns are also allowed:

3 Predicting patient survival times

3.1 User-defined training and validation sets

Once the data is loaded in R and ready for use, it can be used for survival time estimation, and for that one approach is for the user to define the training and validation sets themselves. On doing so one must consider that the training set be big enough to derive a meaningful model from (regression coefficients/weights of the covariates) to be used on the validation set. Typically the training and validation sets should be representative of each other (one shouldn't contain all the long survivors, censored patients, etc., but such features should represent almost the same fraction of the total samples in both cases).
Once the user has defined the patients to be used in the training and validation set (without cross-over) they should plug in the values in the function `STpre-`

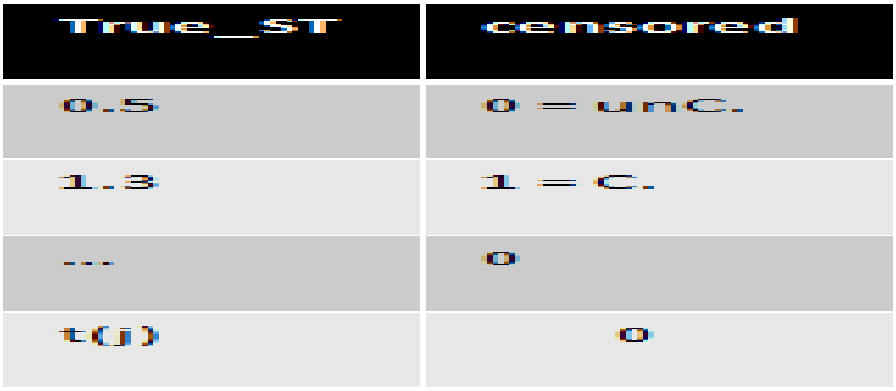

Figure 2: Figure depicting how censorhip data is entered for a patient set

`dictor_BLH`: `STpredictor_BLH`(geDataS, survDataS, cut.off, file, geDataT, survDataT, groups , a , b, q, s, BLHs, geneweights, method , noprior , extras)

This function trains on the data assigned in:

- *geDataT:* gene expression data for training samples
- *survDataT:* survival data for training samples,

and estimates the weights to be assigned to each of the covariates assuming that there is a prior distribution that minimizes the effect of irrelevant covariates. This prior is defined by arguments q and s which are chosen by the user. The choice of the best parameters is usually dependant on the user's experience.

The shape of the prior given the chosen parameters can be visualized by the function `pltprior`(q,s).
For example, the values used in the example stated below (q=1, s=1) define a prior distribution that can be visualized as follows (Figure 3):

```
> pltprior(q=1, s=1)
>
```

Alongside, the model assumes that baseline hazard changes with time, and the user has to define how many different, equally spaced intervals along the time line are expected/thought to have a different value to the others (done in the value assigned to the groups argument) and the optimum values are estimated from the training data. A gamma distribution defined by shape parameter *(a)* and scale

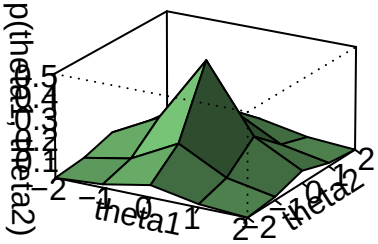

Figure 3: Plot of prior distribution with q=1 and s=1

parameter *(b)* is assumed as a prior on the baseline hazards and must also be assigned by the user.
The shape of this gamma distribution can be visualised using the function `pltgamma` as follows (Figure 4):

```
> pltgamma(a=1.558,b=0.179)
```

Now the arguments relevant to the optimization step in which the best model parameters are chosen according to the training data should be assigned:

- *BLHs, geneweights*: are the starting values for the baseline hazards and the covariate weights defined by the user from which the optimization algorithm is initiated. The length of BLHs should be equal to groups, and the length of geneweights should be equal to the number of columns in geDataS and geDataT.

- *method*: is the optimization method the user picks. It can be any of conjugate gradient descent (CG), BFGS, L-BFGS-B, simulated annealing (SANN).

- *noprior*: defines the number of steps the optimization algorithm is run without a prior on the weights.

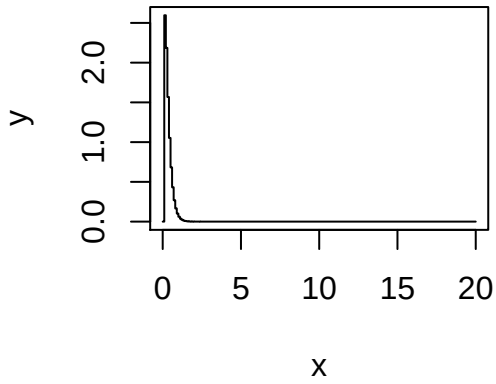

Figure 4: Plot of gamma distribution with a=1.558 and b=0.179

- *extras*: is a list of arguments relevant to the optimization process and corresponds to the *control* argument in the `optim` function in the stats package in R. See `http://sekhon.berkeley.edu/stats/html/optim.html` for details.

Once the predicted survival times are estimated, two groups, long and short survivors are returned by splitting the patients into those who have predicted survival times equal to or greater than the *cut.off* value and those who have a smaller predicted survival time.
Finally the file argument defines the path into which the details and results of this function are stored.
Example:

```
> data(Bergamaschi)
> data(survData)
> result <- STpredictor_BLH(geDataS=Bergamaschi[1:27, 1:2], survDataS=survData[1:
+ survDataT=survData[28:82, 9:10], q = 1, s = 1, a = 1.558, b = 0.179, cut.off=15
+ extras = list(reltol=1))
```

This means that we use the patients 28 : 82 in the Bergamaschi data set for training, and only the covariates 1 and 2 are required as part of the model. Also, only columns 9 and 10 in the survData data frame are really necessary (since they

correspond to censored and True_STs).
The number of intervals thought to have differing baseline hazards are 3, for which the BLHs and geneweights arguments are left out since they are assumed to be equal to the default NULL. This way, the function assumes the geneweights are all initially zero, and the BLHs the mean of the gamma distribution as defined by a and b, which are defined by the user to be 1.558 and 0.179 respectively. The method chosen is conjugate gradient decent (CG) and the optimization is only performed once without a prior to prevent the algorithm from terminating at the peaked origin. The relative tolerance is set to 1 in extras.
According to the predicted survival times determined on the patients 1 : 27 defined in geDataS and survDataS, the user requests that the patients be split into long and short survivors around the value of 3 years as defined in cut.off.

3.2 k-times cross validation

Another possibility is to perform a k-times cross validation on the entire data set instead of splitting the data beforehand into training and validation sets. In this case the user would use the function `STpredictor_xvBLH`:
`STpredictor_xvBLH`(geData, survData, k , cut.off , q, s, a , b, groups, method , noprior, extras),
which is very similar to the previous function except that we do not define a training and validation set, but the whole data set we will be using as *geData* and *survData* and the number of partitions the cross-validation is performed on is given by *k*, while all other arguments apply as discussed in the previous section.

4 Accessing the weights and baseline hazards returned

4.1 STpredictor_BLH

The weights and baseline hazards returned from this function can be accessed in the "log_optimization" item of the list of returned results. In the example we mentioned before the results were assigned as result and so we can access them by typing `result$log_optimization$par`. The first "group" components correspond to the baseline hazards, while the rest correspond to the weights of the covariates.

4.2 STpredictor_BLHxv

The weights can be accessed as "weights" in the list of results, and the baseline hazards as "baselineHs" in the results. For example, if the results of the function

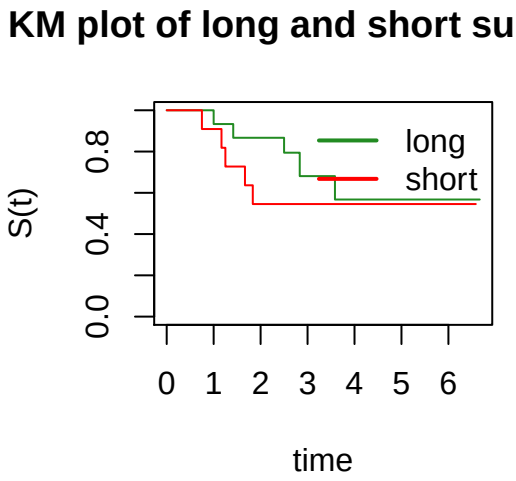

Figure 5: Kaplan-Meier plot of the long and short survivors

are stored as "result", the weights can be accessed as `result$weights`, and the baseline hazards as `result$baselineHs`.

5 Plotting the KM curve of short and long survivors in the same plot

To compare the survival curves of the short and long survivors returned from either of `STpredictor_BLH` or `STpredictor_xvBLH` the user can use the function `kmplt_svrl` which takes the following arguments:
`kmplt_svrl`(long, short, title)
In this case the arguments `long` and `short` correspond to the long and short survivor groups to be plotted. In any case, these are data frames that must contain a column with the "True_STs" and "censored" columns. On visualizing the results as returned from the two functions in the package the function is used as follows (Figure 5):

```
> kmplt_svrl(long=result$long_survivors, short=result$short_survivors,title="KM p
```

In case the user also wants to plot the survivor function of the entire data set for sake of comparison, another argument "all" can be added, where the data frame containing the survival data for the entire data set is passed.

6 Plotting the receiver-operator curves at a particular time point

This is done using the `survivROC` function which takes the following arguments:
`survivROC`(Stime, status, marker, predict.time, cut.values, plot), where,

- *Stime*: is the true survival times of the patient, which in our case (according to the example in STpredictor) is `result$predicted_STs$True_STs`.
- *status*: denotes whether the patient is censored or not. According to our example it can be given by `result$predicted_STs$censored`
- *marker*: requires the predicted survival times, accessed according to our example by `result$predicted_STs$Predicted_STs`.
- *predicted.time*: gives the time point at which the ROC curve is to be constructed.
- *cut.values*: are the "cut.off" values at which the sensitivity and specificity are calculated for constructing the ROC curve. It is a vector with all values, or can be set to NULL and automatically assigned by the software.
- *plot*: a logical variable, where T indicates that the plot should be generated and F that it should not.

7 Generating a plot of the areas under all ROC curves and calculating the area under it

This is usually used as the most appropriate measure of predictor performance, unbiased by the choice of cut-off value or the change in the population with time and can be done by calling the function `survivAURC` which takes the following arguments:
`survivAURC`(Stime, status, marker, time.max)
where *Stime*, *status* and *marker* are the same as in `survivROC` function and `time.max` is the maximum time at which the ROC curves are estimated. For example, according to the example given for the function `STpredictor_BLH`, the plot of all the Areas under the ROC curves generated up till time point 20 years with a resolution of 1 year in between is given by as seen in Figure 6:

```
> survivAURC(Stime=result$predicted_STs$True_STs,status=result$predicted_STs$cens
```

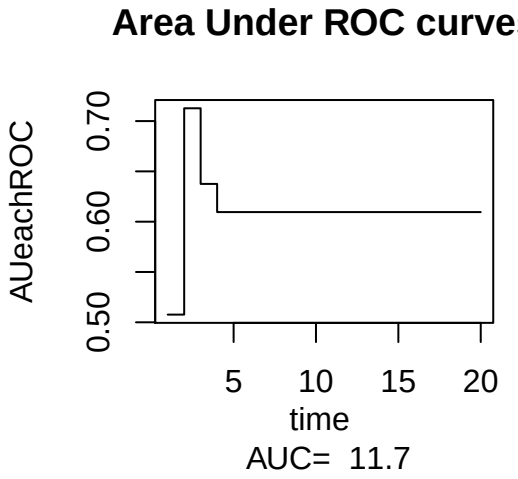

Figure 6: AURC curve

```
$AUC
[1] 11.66154

$AUeachROC
 [1] 0.5076426 0.7127124 0.6375423
 [4] 0.6095430 0.6095430 0.6095430
 [7] 0.6095430 0.6095430 0.6095430
[10] 0.6095430 0.6095430 0.6095430
[13] 0.6095430 0.6095430 0.6095430
[16] 0.6095430 0.6095430 0.6095430
[19] 0.6095430 0.6095430
```

8 Comparing two populations according to their KM survival curves

As shown in `kmplot_svrl`, the user can visualize how the plots of two patient populations compare in a single plot. When wanting to determine how statistically significant the difference between these two populations is, the user can perform a log rank test by calling the `log_rnk` function. The arguments passed are simply: `logrnk`(dataL, dataS), where

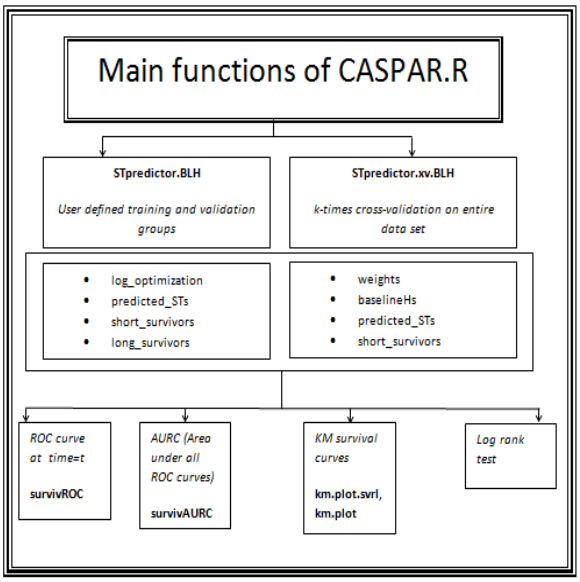

Figure 7: An overview of the functions of RCASPAR and how they feed into each other.

- *dataL*: is the dataframe containing at least the true survival times (True_STs) and censorship status (censored) of the long survivors (group 1) as well as a column "group" with entries "L" denoting that they belong to the long survivor group.
- *dataS*: is a similar dataframe with the data relevant to the short survivors where the entries in "group" are "S" instead of "L".

Following the example we have been using so far, we can do this for the short and long survivors returned from the previous example as follows:

```
> logrnk(dataL=result$long_survivors, dataS=result$short_survivors)

$Xsq
[1] 0.7582287

$pValue
[1] 0.3838834
```